REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*
08-01-2004

2. REPORT TYPE
Final Report

3. DATES COVERED *(From - To)*
June 20, 2001 to May 31, 2003

4. TITLE AND SUBTITLE
Power Electronics Building Blocks "Plug and Play"
Hardware and Software Control Architectures

5a. CONTRACT NUMBER
Award No. N00014-01-1-0954

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Dr. Dushan Boroyevich

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
OFFICE OF SPONSORED PROGRAMS
301 BURRUUS HALL
BLACKSBURG, VA, 24061-0249

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
OFFICE OF NAVAL RESEARCH
BALLSTON CENTRE TOWER ONE
800 NORTH QUINCY STREET
ARLINGTON, VA 22217-5660

10. SPONSOR/MONITOR'S ACRONYM(S)
ONR

11. SPONSORING/MONITORING AGENCY REPORT NUMBER

12. DISTRIBUTION AVAILABILITY STATEMENT

"Approved for Public Release; distribution is Unlimited"

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The main objective of this project has been to investigate means to standardize communications and control systems in order to develop seamless “Plug and Play” (PnP) power electronics. The intent within has been to pave the way for the development of reconfigurable low-cost, high reliability, and easy to use power processing devices. Such devices, known as Power Electronics Building Blocks (PEBBs), would certainly encourage the proliferation of power electronics into markets not yet penetrated due to a significant, critical lack of industrial modularization and standardization in this area. In fact, the flexibility level that could be attained is such that it would ensure significant increments of production, as well as manufacturing cost reductions due to economies of scale.

15. SUBJECT TERMS
Power Electronics Building Blocks (PEBB), Static Power Converters, Control Software Architecture, Distributed Control Architectures.

16. SECURITY CLASSIFICATION OF:

a. REPORT	b. ABSTRACT	c. THIS PAGE	17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
				231

19a. NAME OF RESPONSIBLE PERSON
Dr. Dushan Boroyevich

19b. TELEPONE NUMBER ***(Include area code)***
(540) 231-4381

CPES
Center for Power Electronics Systems

Power Electronics Building Blocks

"PLUG AND PLAY"

Hardware and Software Control Architectures

FINAL REPORT

***Prepared for*:**

Office of Naval Research

Dushan Boroyevich, and Stephen Edwards

Sumithra Bhakthavatsalam, Rolando Burgos, Jerry Francis, Daniel Ghizoni, Jinghong Guo, Konstantin Louganski, Xiangfei Ma, Parool Mody, Sebastian Rosado, Wei Shen, Kuljeet Singh, and Luca Solero

Contract Information

Contract Number	N000140010489
Title of Research	PEBB Plug and Play
Principal Investigator	Dushan Boroyevich
Organization	Center for Power Electronics System at Virginia Tech

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 EXECUTIVE SUMMARY

1.1 Introduction

POWER ELECTRONICS BUILDING BLOCKS, "PLUG AND PLAY" HARDWARE AND CONTROL ARCHITECTURES. The main objective of this project has been to investigate means to standardize communications and control systems in order to develop seamless "Plug and Play" (PnP) power electronics. The intent within has been to pave the way for the development of reconfigurable low-cost, high reliability, and easy to use power processing devices. Such devices, known as Power Electronics Building Blocks (PEBBs), would certainly encourage the proliferation of power electronics into markets not yet penetrated due to a significant, critical lack of industrial modularization and standardization in this area. In fact, the flexibility level that could be attained is such that it would ensure significant increments of production, as well as manufacturing cost reductions due to economies of scale.

To this end, a PEBB-based distributed power electronics system architecture was proposed, built, and evaluated. This system is depicted in Fig. 1-1, where it clearly shows the hierarchical structure of the developed control system, as well as its main constituents, i.e. the Universal or application Controller, and the PEBB modules with their Hardware Managers. Fig. 1-2 on the other hand shows the information flow in the proposed system, where control signals, together with state variables measurements and various commands are transmitted through and across hierarchies using the communications protocol PESNet. This protocol has also been developed in this project and implemented by means of a double-ring structure fiber optic network. Implicit in this figure is the software and control software architecture which enables the PnP capabilities of this system, exuding in modularity, reconfigurability, and reusability.

The specific structure and partitioning of the proposed power electronics system was determined through thorough studies of power conversion systems. Particularly, voltage-source-based power converter structures were considered, representing if not all high power structures all high power applications. From these, a common element was identified and through appropriate correspondence with the control system structure a PEBB module was defined. Particularly, the PEBB was defined as a converter phase-leg, functionally equivalent to a Single-Pole-Double-

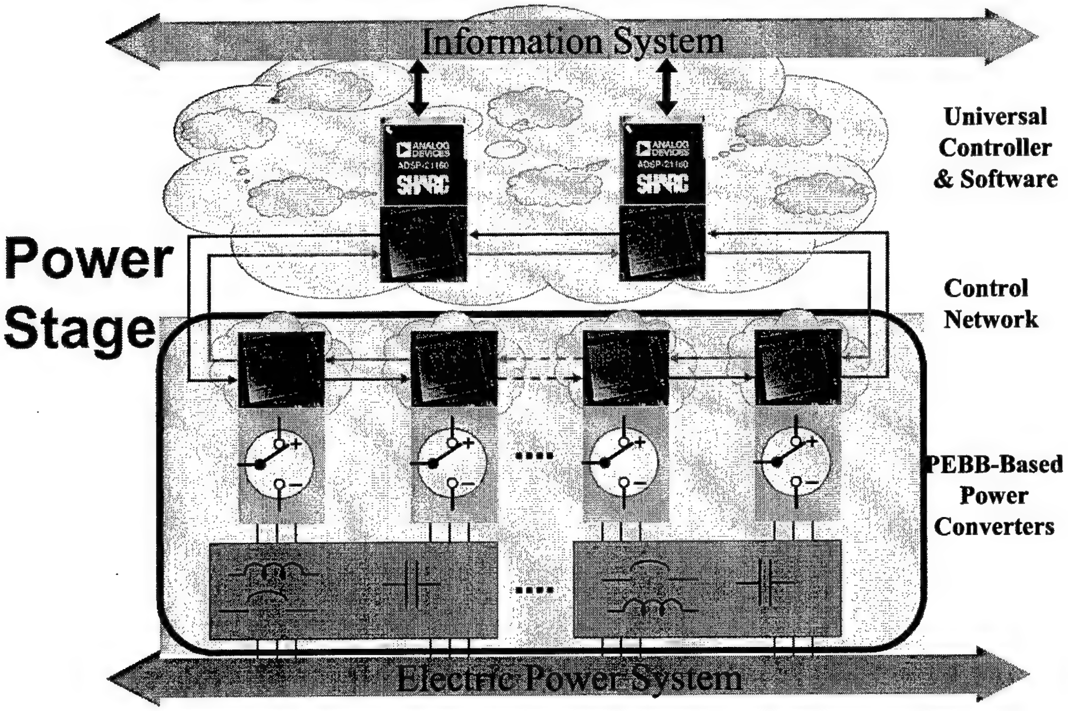

Information System
ANALOG DEVICES ADSP-21160 SHARC
ANALOG DEVICES ADSP-21160 SHARC
Universal Controller & Software
Power Stage
Control Network
PEBB-Based Power Converters
Electric Power System

Fig. 1-1 PEBB-based distributed power electronics system.

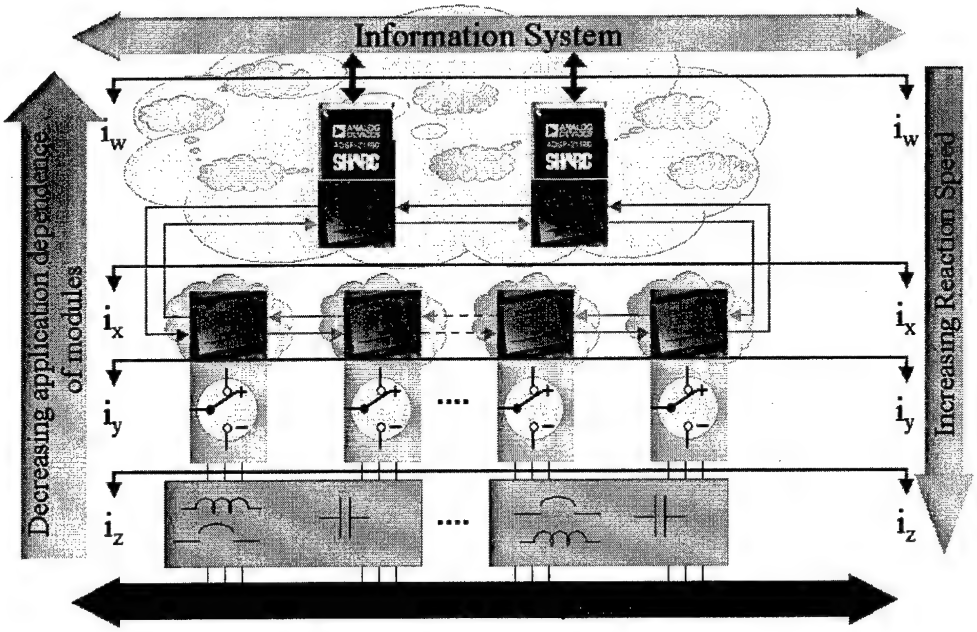

Information System
ANALOG DEVICES ADSP-21160 SHARC
ANALOG DEVICES ADSP-21160 SHARC
Decreasing application dependence of modules
Increasing Reaction Speed
i_w
i_x
i_y
i_z

Fig. 1-2 Information flow throughout the power electronics system.

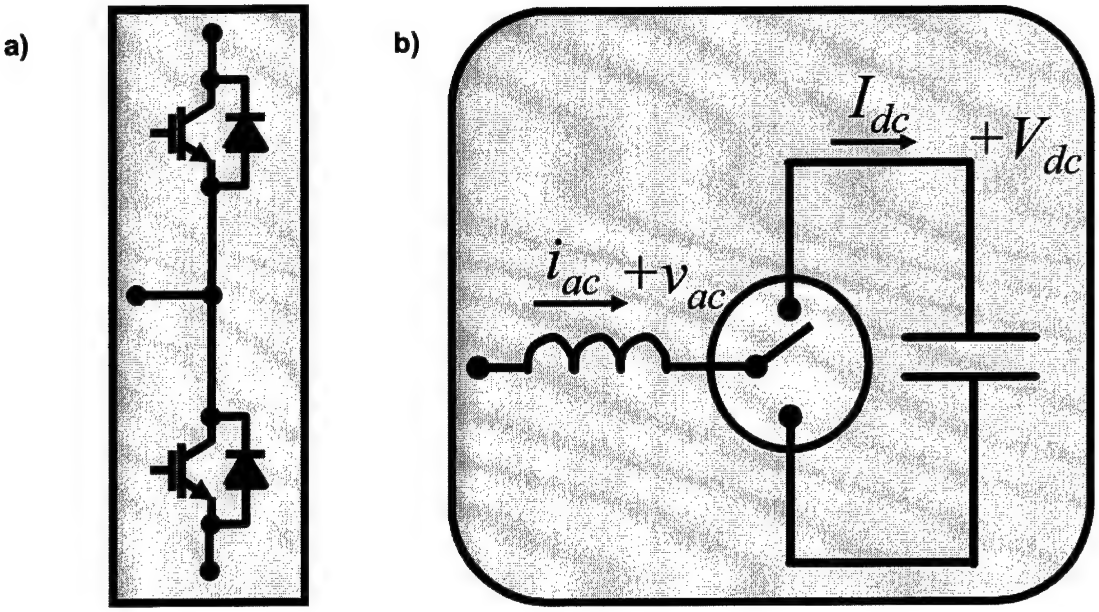


Fig. 1-3 PEBB module defined for this project. a) Circuit schematic and b) functional equivalent modeled as a SPDT switch.

Throw (SPDT) switch as shown in Fig. 1-3. Fig. 1-4 shows how this PEBB is found in most voltage-source power converter structures employed for medium to high power applications.

The final implementation of the proposed power electronics system is depicted in Fig. 1-5, where again the main constituents can be easily identified, as well as secondary components which nonetheless represent critical system functions. Specifically, the electromechanical structure which must accommodate the new distributed power electronics system, which must cope with, solve, and minimize space distributions and orientations, unwanted parasitic effects, all the while providing the main PnP functionalities, that is ease of reconfiguration, maintainability, supportability, and modularity. Fig. 1-5 also shows what have been defined as passive PEBBs. These are mainly reactive elements required for proper converter operation and harmonic and EMI filtering, and protection and operating devices such as fuses and contactors. This project did not consider fully integrating these passive PEBBs, nonetheless they have been physically located and distributed accordingly, and have been provided with integrated protective devices.

As previously stated, the main drive of this project has been to investigate means to standardize control and communication systems in order to develop PnP power conversion systems. Consequently, the main work was focused in the development of 1) the Universal

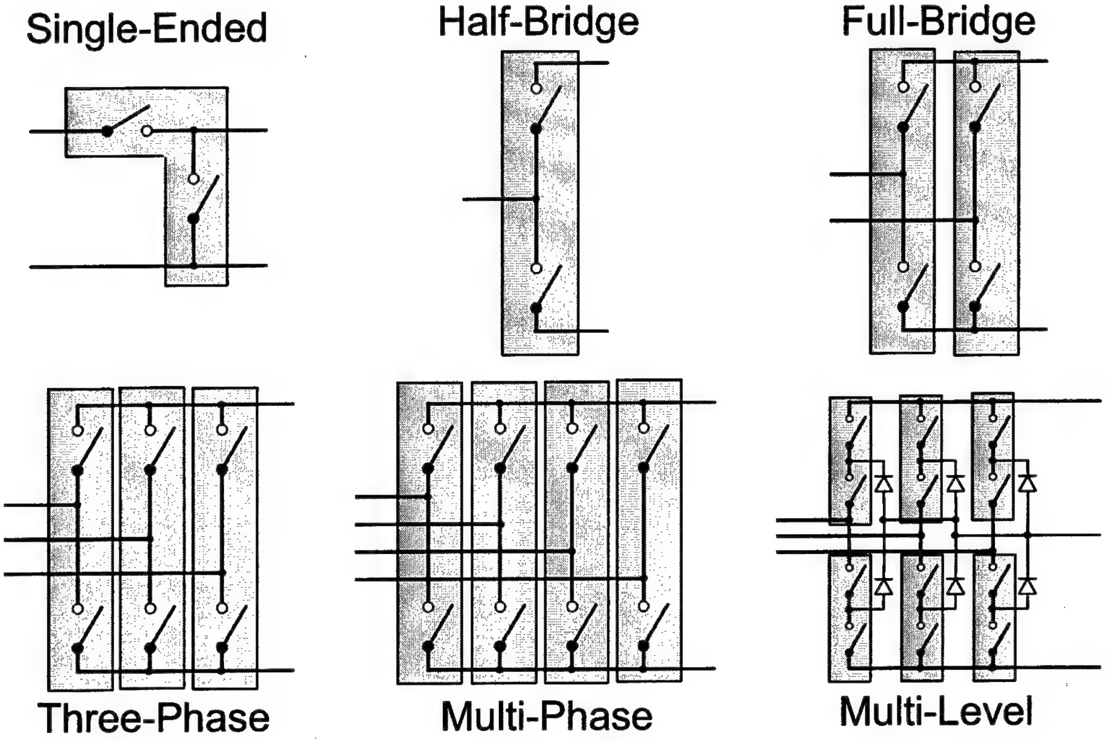


Fig. 1-4 Voltage-source power converter topologies with highlighted PEBB modules showing the existence of a common element between all of them.

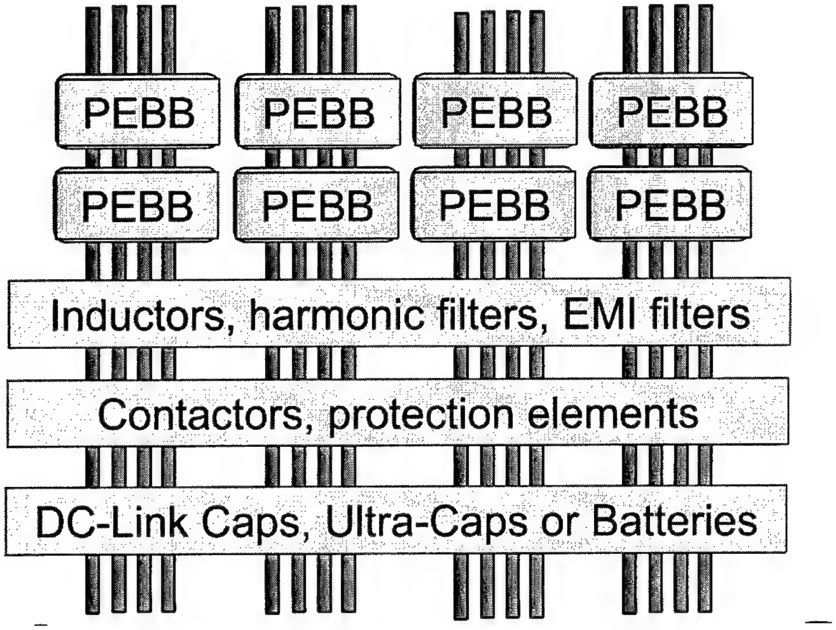
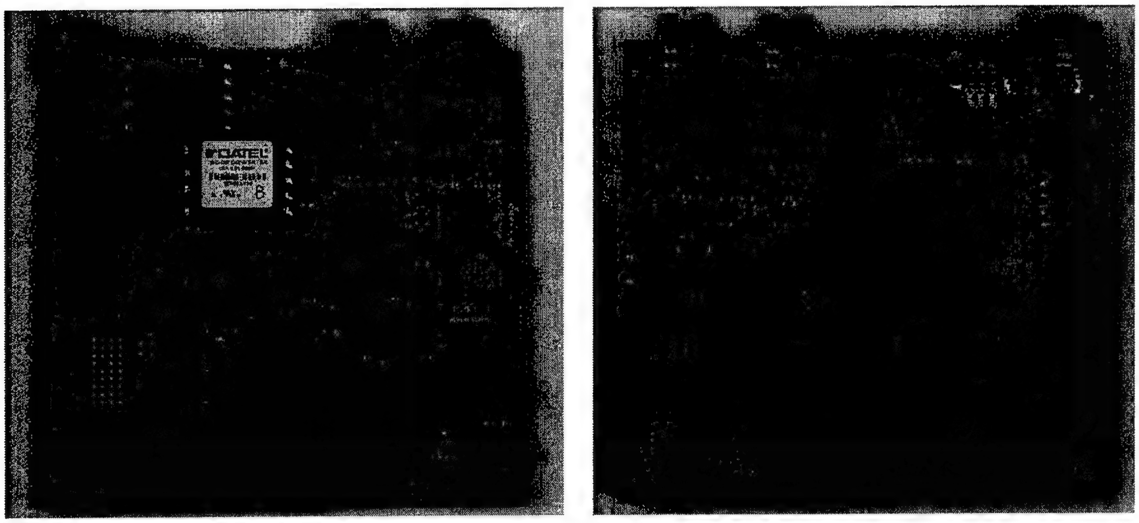

PEBB
PEBB
PEBB
PEBB
PEBB
PEBB
PEBB
PEBB
Inductors, harmonic filters, EMI filters
Contactors, protection elements
DC-Link Caps, Ultra-Caps or Batteries

Fig. 1-5 Physical space distribution of components for the developed PEBB-based power electronics system.



FRONT *BACK*

Fig. 1-6 Universal Controller board developed and manufactured for this project.

Controller, 2) the communications and control software architectures, and 3) the Hardware Manager.

- It can be said that the Universal Controller effectively achieved all its design goals, offering powerful computational resources as well as a several communication interfaces for ease of development, operation, and interaction. Built over a DSP/FPGA digital system architecture, the Universal Controller now includes interfaces such as: JTAG for the DSP and FPGA, 88 I/O pins connected to the FPGA, and a PCI interface. Fig. 1-6 shows a picture of the Universal Controller.

- The PESNet protocol based on industrial control system protocols was further upgraded to include enhanced communication capabilities to support the PnP capabilities of the proposed power conversion system implemented over a dual ring fiber optic network. On the other hand a control software system was built over standardized Elementary Control Objects (ECO) featuring a high degree of reusability and great ease of reconfiguration for different functions and applications. All this was developed over an especially designed and developed kernel DARK, which provided a high performance platform for running dataflow applications. Fig. 1-7 shows a closed-loop control implementation using the proposed control software architecture.

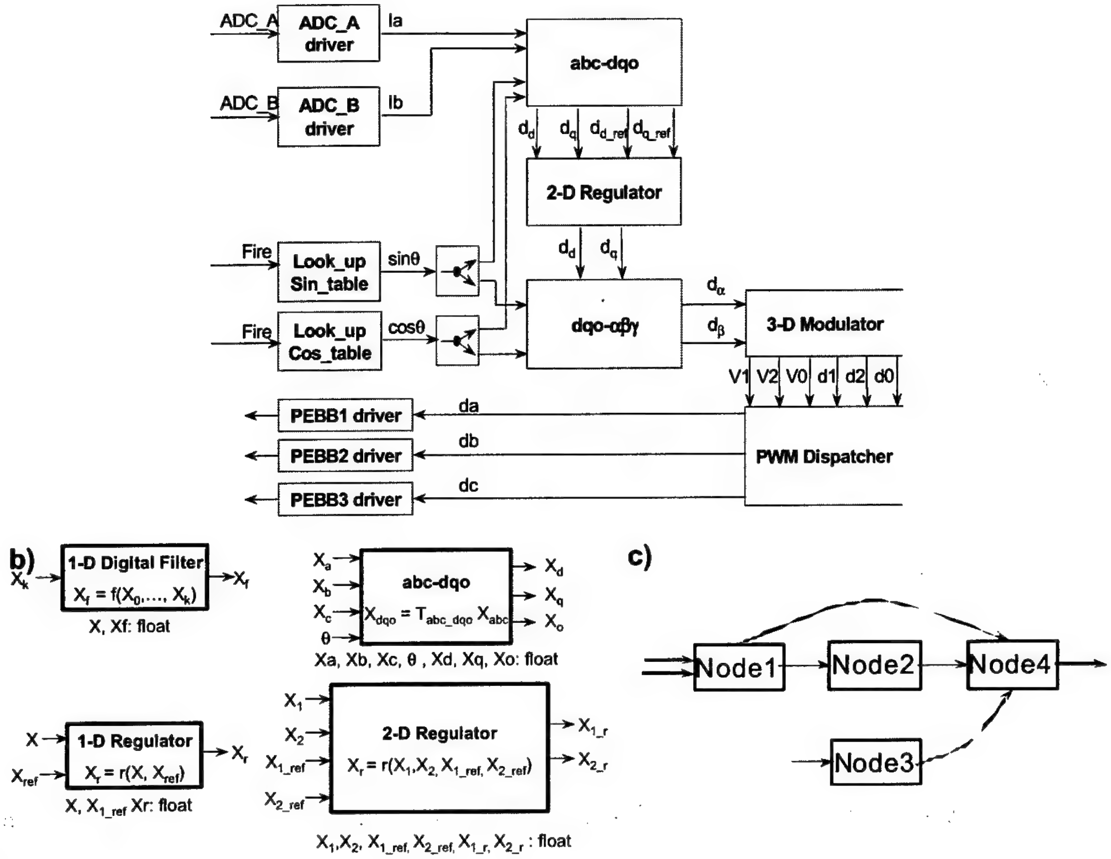


Fig. 1-7 a) Closed-loop control block diagram of a three-phase voltage-source inverter using elementary control objects (ECO), b) Data flow architecture, and c) ECO.

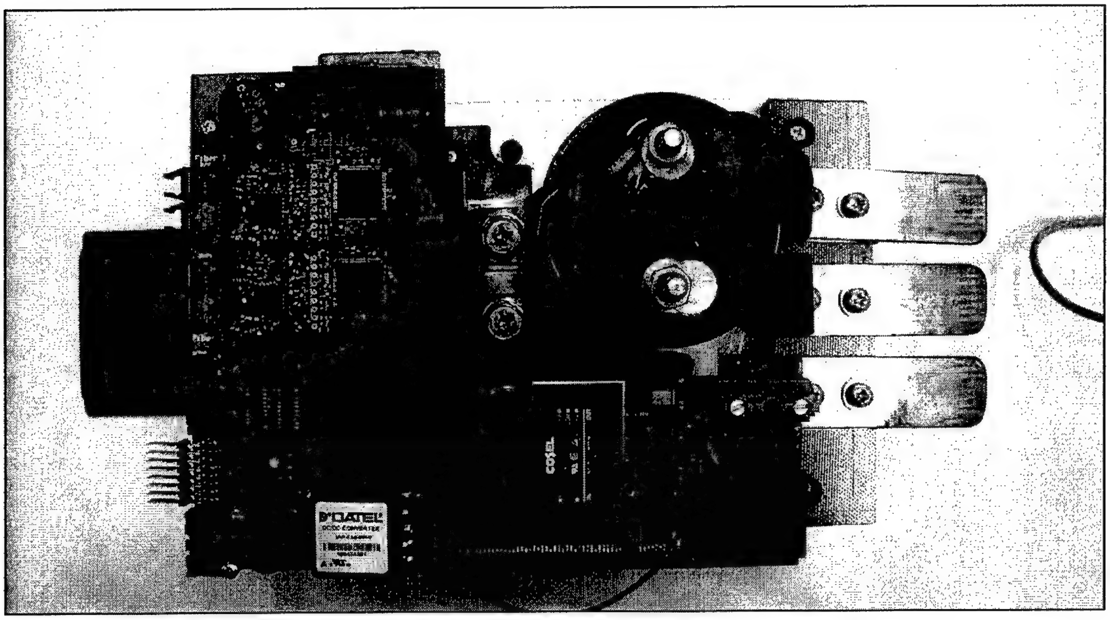


Fig. 1-8 Hardware Manager board mounted on one of the 33 kW PEBB modules.

- The Hardware Manager, depicted together with a PEBB module in Fig. 1-8, was designed to control the newly developed 33kW PEBB modules. Built over a Xilinx FPGA, it supports the dual-ring-based PESNet communications protocol, and features several built-in protection devices and debugging features for aiding in the system design. Its new design exploits all the capabilities provided by VHDL coding, presenting great programming and reconfiguration capacities.

A brief description of the main achievements for each of the above mentioned topics is provided in the sections hereinafter.

1.2 Universal Controller

The main goals for developing a Universal Controller for PnP PEBB-based power electronics systems have been firstly to reduce their design cycle time and cost, secondly to increase their flexibility, modularity, and reconfigurability, thirdly to increase their reliability, supportability, and maintainability; and fourthly to increase their overall capacities and capabilities. As shown in our previous work on PnP PEBB-based power electronics systems, distributed control architectures at converter level makes these systems open, flexible, and simple to use. The Universal Controller plays a key role in this scheme, as it is in charge of performing all high-level application-oriented tasks. For this purpose, this controller was designed to offer enhanced and more powerful computational resources as well as a variety of communication interfaces, and was built over a DSP/FPGA digital system architecture. As such, the Universal Controller is a main constituent of the PnP PEBB-based power electronics system structure proposed and developed in this project.

From the goals aforementioned, the design cycle of a power converter is one of considerable importance given its implications on development and production costs. This cycle time is highly affected by the need to develop a new control system for every application, reason why the usage of a Universal Controller becomes apparently attractive. This allure is further increased when standardized interfaces are defined and established between all the system components. This project supports this goal by including the following interfaces to aid in the design of the control system, namely JTAG for the DSP and FPGA, 88 I/O pins connected to the FPGA, and a PCI interface. These I/O pins have a functional purpose after debugging, but they proved to be extremely useful during this stage as well, as they added excellent visibility into the FPGA code during runtime. The PCI interface on the other hand provided great visibility over the whole system by easily enabling the designer to monitor and access variables while the controller is running. This particular interfacing mode is still under development in collaboration with Northrop Grumman Corp.

A key factor for a distributed power electronics systems is the actual communication between its components. This was addressed in this project by developing and upgrading PESNet, a protocol based on the industrial protocol MACRO. Such a network has the advantage of allowing for all components to be accessed and programmed through a single device, the Universal Controller. Also, PESNet employs a dual-ring fiber-optic network structure, which greatly increases the overall system reliability by providing dual and backup paths for accessing each component.

1.3 Control Software Architecture

It has been empirically shown that the software architecture employed on PnP PEBB-based power electronics systems is just as important as the actual algorithms and data structures employed. On this matter, this project continued our previous work on software for embedded control systems, and proposed a power electronics control software built over standardized modules -Elementary Control Objects (ECO)- having a high degree of reusability and reconfigurability. For this purpose, a kernel dubbed DARK was especially designed, evaluated, and successfully compared with commercially available kernels for industrial control systems. DARK met the goal of providing a high performance platform for running dataflow applications as in the proposed embedded control for power electronics applications.

Further improvements for this embedded control kernel were attained by programming it in C++, which provided a better and more structured way of extending data channels for user-defined data structures. Also, a distributed, transparent messaging protocol for PnP PEBB-based power electronics systems was designed, which allowed for transparent messaging between controllers across a multi-controller application network. Finally, in order to assess the effectiveness and feasibility of the proposed and developed control, kernel, and protocol system several commercial software packages were investigated. Though the commercial software presented a friendlier user interface through the use of a graphic development environment, the proposed system provided more flexible real-time control options, eased the design of distributed control systems, and required significant less redesign efforts.

1.4 Hardware Manager

The proposed PnP PEBB-based power electronics system architecture has as main power device controller the Hardware Manager, or simply put, the power stage controller. This controller is basically responsible for low level hardware oriented tasks. It is intrinsically application-blind, it actuates over its power components, and receives commands –from a Universal Controller- and transmits gathered information from its restricted system, i.e. values of current, voltage, temperature etc. Such a controller was completely designed, manufactured and successfully tested in this project. This new Hardware Manager was designed as part of the new 33kW PEBBs also built in this project for testing and verification purposes. The new Hardware Manager employs new technology, making it fully compatible with the Universal Controller and capable of fully implementing the newly developed PESNet protocol with enhanced capabilities.

It is built over a Xilinx FPGA, supports the dual-ring communications network in use, has built-in protection devices, as well as several debugging features for aiding in its design.

On a board level, the Hardware Manager design was done exploiting all the experience attained with the Universal Controller. Following a PEBB modular approach, every functional part of it, from communications to sensors, is basically independent of each other. In fact, some subsystems have the exact same circuit structure as in the Universal Controller. Software-wise, the VHDL code has been individually addressed per functions, which significantly simplifies any programming and troubleshooting required when modifying its code. Also, its communications capabilities were proven to every expectation, and showed an excellent performance, as well as the sensors operation. The Hardware Manager has sensors for measuring the power stage current and voltage, as well as the on-board temperature. This last reading is used as a protection device together with an over-current signal generated by the IGBT IPM itself. In all, the new PEBB/Hardware Manager has proven all its capabilities and usefulness for PnP PEBB-based power electronics systems.

1.5 PEBB-Based Power Stage

Up to the appearance of the PEBB concept for power electronics systems, medium to high power static power converters were mostly designed and manufactured on a customized, per application basis. For tailored designs as these ones significant optimizations in electrical, magnetic, and mechanical systems may be achieved, thus minimizing unwanted effects such as losses, parasitic inductances, capacitances, EMI, resonances etc. A PEBB-based system however lacks this optimization capability in the sense that it must provide for unparalleled modularity and reconfigurability, the very essence of its existence. In fact, a PEBB-based system is built having a variable space distribution, ideally an unbounded one which allows for easy reconfiguration, modularity and scalability. Consequently, these systems must deal with all parasitic effects associated to power electronics in order to operate successfully and become a feasible alternative.

It is evident then that in order to minimize these obstacles intrinsic to PEBB-based systems a different design approach is required. Particularly, it now becomes desirable to design a distributed power stage structure in order to minimize all the aforementioned unwanted effects. A first step was hence taken in this project by performing a partitioning study, which determined both physical and information boundaries wherefrom to define PEBB modules in power electronics systems. From these analyses feasible power stage partitioning criteria were defined, as well as information layers and communication and data channels applicable to distributed control systems. Correspondingly, a power stage system was designed based on the temporal and physical distributions and partitions previously defined. New PEBB modules using the newly designed Hardware Manager were designed and manufactured, previously built PEBB modules were upgraded to accommodate for soft switching capabilities, and an appropriate cabinet structure supporting all the PEBB concept capabilities was built. All of these were successfully tested so far.

From the work performed in this project, future improvements and actual modifications that will be necessary to make the electrical system structure fully compatible with the PEBB concept have been determined. Specifically, there is an apparent need for optimized design for busbars and busplanes for the DC bus. There's also a need for defining active and passive PEBBs, and for designing, specifying, and standardizing connectors, protection devices, low power distribution devices, etc.

2 Universal Controller

The Universal Controller is used to control PEBB modules. It is power level independent, and controls other modules via a fiber optic network. The controller has been through two revisions, and nine revised controllers were made based on issues found in the first revision. Fig. 2-1 and Fig. 2-2 show the universal controller.

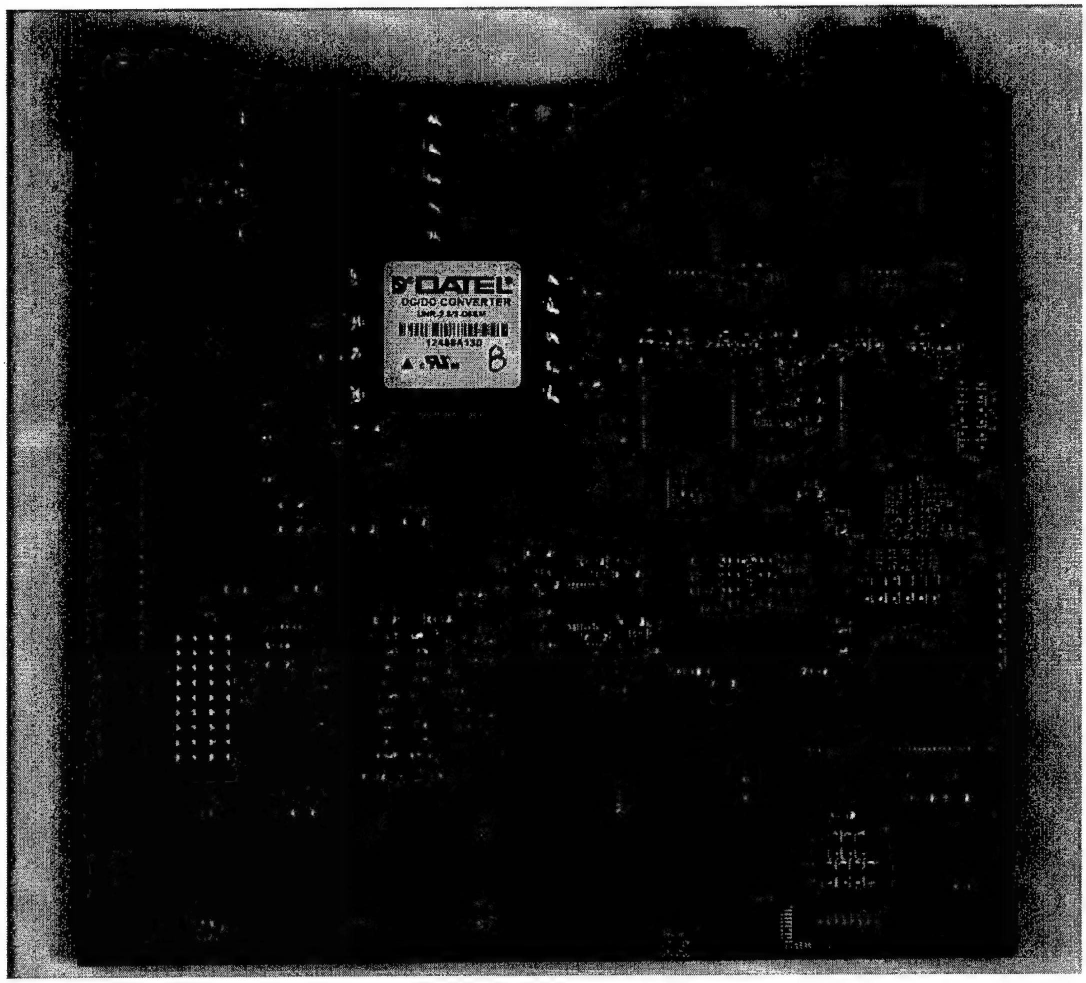


Fig. 2-1 Universal Controller Front

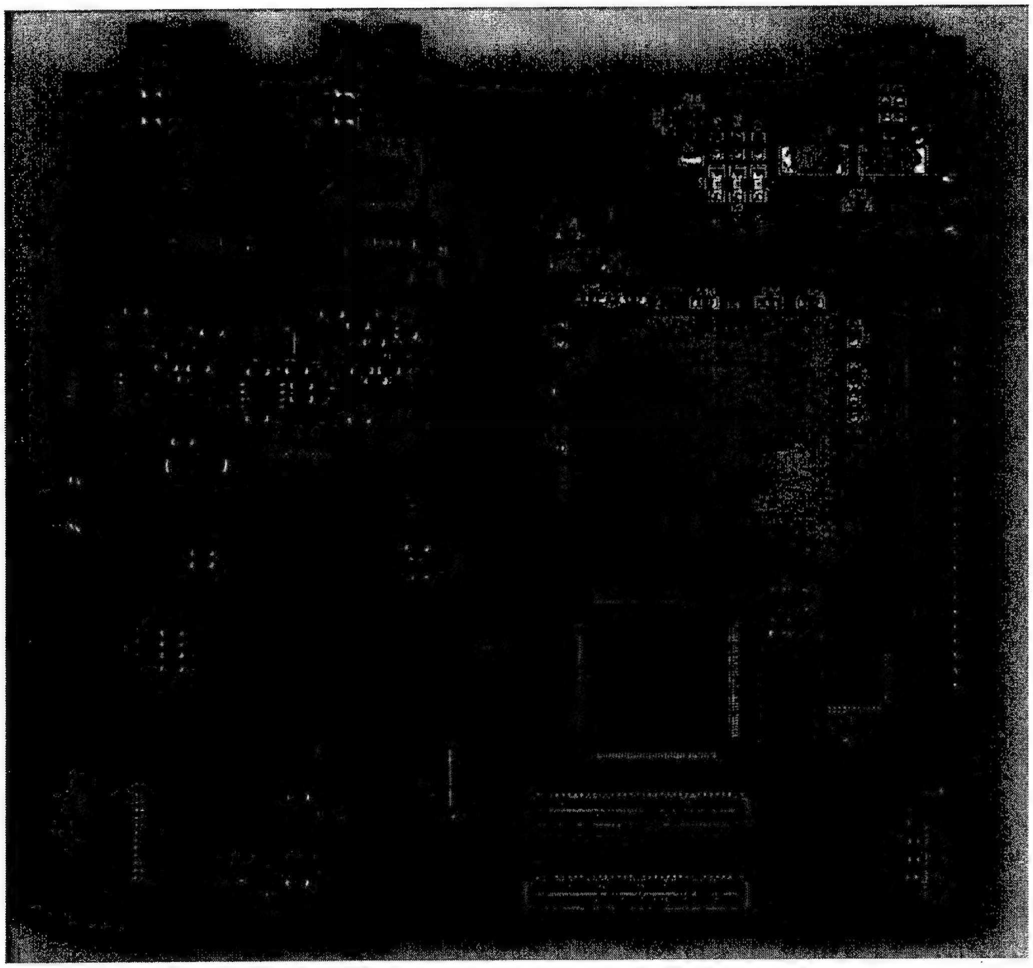


Fig. 2-2 Universal Controller Back

The universal controller has been designed to address the needs of the majority of power electronics applications in medium to large power electronics converters. Large systems need communication interfaces, status indicators, debugging tools, advanced control algorithms and fault tolerance. We kept this in mind when designing the controller. Also, in large systems components of converters can be distributed across large areas. For that purpose we have developed a communication protocol called PESNet, which allows these modules to be connected together in a fault tolerant fiber optic ring. This network structure was partially used in the preceding project, as it was just a single ring and hence not fault tolerant.

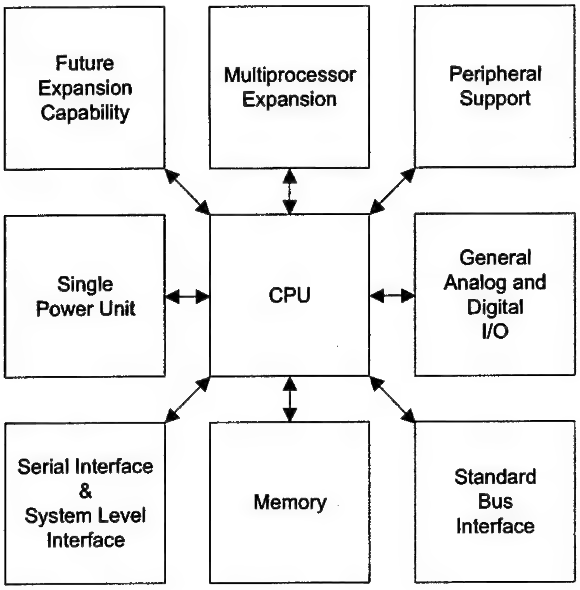

Future Expansion Capability
Multiprocessor Expansion
Peripheral Support
Single Power Unit
CPU
General Analog and Digital I/O
Serial Interface & System Level Interface
Memory
Standard Bus Interface

Fig. 2-3 Controller Functional Block Diagram

2.1 Specifications

There are several features desirable in a distributed controller. The most obvious is the need for a CPU to execute code written to control the application. The controller should be able to communicate with the external world in which it exists. This is done via standardized system interfaces. The standardized interfaces allow the controller to be quickly integrated into a system without having to design customized interfaces and support. Scalability is also a desirable property. Applications will require more than one controller. There should be a way for multiple controllers to work together to solve large problems. It is impossible to predict and implement every requirement for every application. There should be some way to expand the controller capability for unpredictable future needs. A block diagram of the fundamental controller properties is shown in Fig. 2-3.

It is desirable to have a modular approach to implementation so that each module can be developed and tested incrementally. Once one module is debugged, it should remain functional while other modules are being developed. It should also be possible to deactivate one module if it is not required in the design. Also, being universal means that the controller should be portable across applications. The strategy to implement the controller should support reconfiguration to allow this.

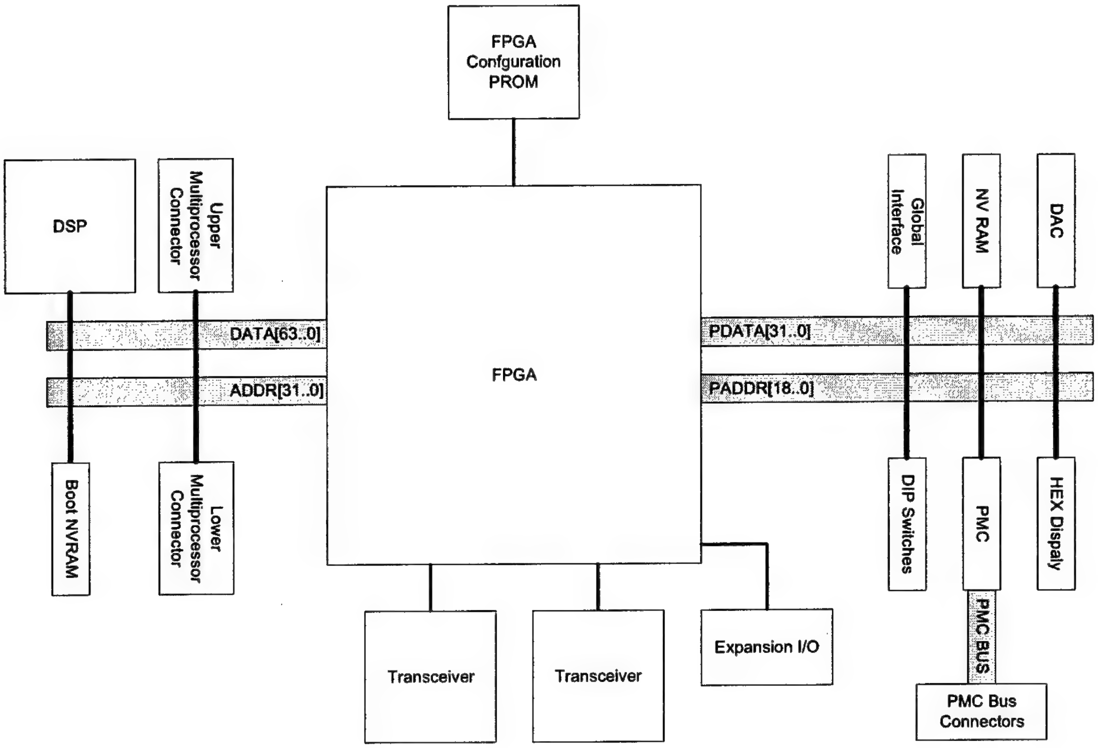


Fig. 2-4 Universal Controller Architecture

2.2 Approach

2.2.1 Architecture

The controller architecture consists of two main busses bridged by the FPGA. This approach allows for incremental debugging of the controller without having to worry about every block at once. This also allows for future architectures in which the FPGA could interact with the peripherals without having to know what the DSP is doing. An alternative would be to have a singe bus with every peripheral connected to it. The controller architecture is shown in Fig. 2-4.

There were two approaches possible when implementing the FPGA control code. The first approach allowed the thread of execution to pass from the DSP, through the FPGA, and into to the ASIC, which would service the request, return control to the FPGA, which would in turn return control to the DSP.

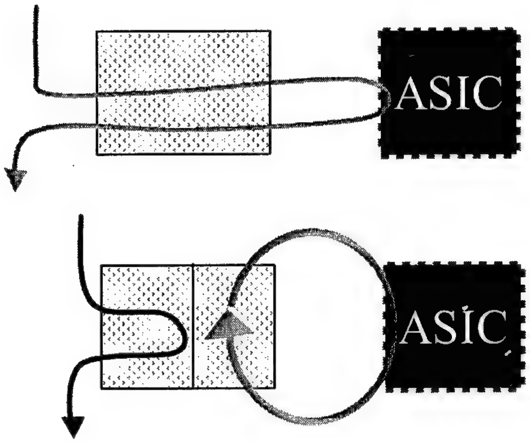


Fig. 2-5 FPGA Control Strategies

The second approach is an asynchronous approach, where the DSP would interface with a set of control registers in the FPGA that would perform a specific task, most likely involving an ASIC. This architecture would require the FPGA to implement a command processor that would execute the commands placed in the command buffer in the FPGA by the DSP, thus creating a second thread of execution. The second thread of execution would need to synchronize with the first one in some way. This complicates the interface between the DSP and the FPGA. It would also mean that after the write to the FPGA returns, there would be no direct way to tell if the operation has completed or not. Reads from ASICs would also be complicated, as the data is not immediately available in the FPGA, and so there would first have to be a request, and then later, the data would become available after the FPGA had retrieved it. These two approaches are illustrated in Fig. 2-5.

As more and more control blocks were implemented, the maximum clock speed of the FPGA started to drop very fast. Eventually, it became difficult to manage each block due to the increased propagation delay within the FPGA. Each control block would have to have address, data, and control lines. Most blocks passed data directly from the DSP to the FPGA directly instead of manipulating the data. Taking this into account, a new architecture was developed that allowed the control blocks to control the data flow while not actually "seeing" the data themselves. Control data that is stored in the FPGA to set some control line, such as the blanking control for the hex display goes to a set of registers. This new approach is shown in Fig. 2-6.

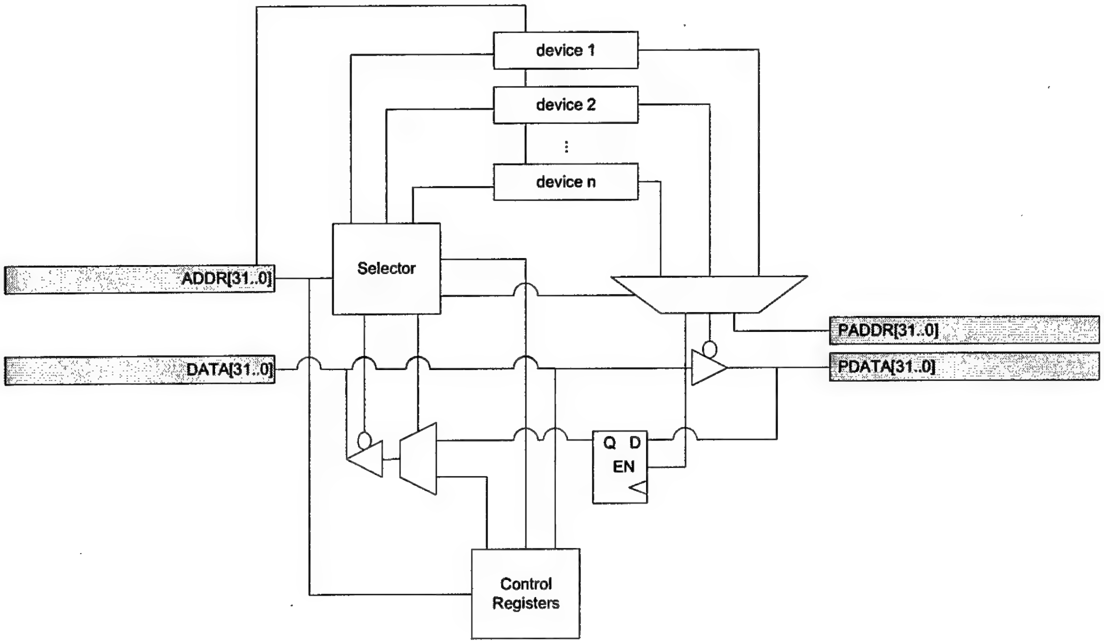


Fig. 2-6 Improved Bus Management

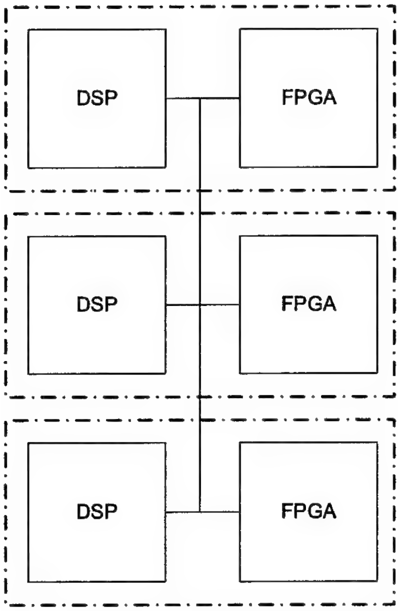


Fig. 2-7 Stacked

In this architecture, several controllers can be connected together for tasks that require larger processing power. When the controllers are stacked, the DSPs and FPGAs for all boards appear in parallel with each other as shown in Fig. 2-7. The FPGA uses the ID of the DSP as part of its address scheme. It is therefore possible for a DSP to access a resource on the FPGA of another board as easily as it would access a resource on one of its own.

2.3 Block Descriptions

2.3.1 DSP

The DSP chosen is an Analog Devices ADSP-21160 80 MHz floating point DSP. The DSP has the following features that make it attractive for use in advanced control architectures:

- Single Instruction, Multiple Data execution
- Intrinsic support for multi-processing
- Built-in multiply and accumulator, barrel shifter, and ALU
- Pipelined execution and instruction loading via Data and Address Generators
- Host Port Interface
- 2 Synchronous Serial Ports
- Built-in control of external port

The FPGA supports the DSP's access to other peripherals on the controller. The DSP interface has been designed according to the architecture described in ###. The DSP communicates with the FPGA using the data bus, the address bus, and some control signals as shown in Fig. 2-8.

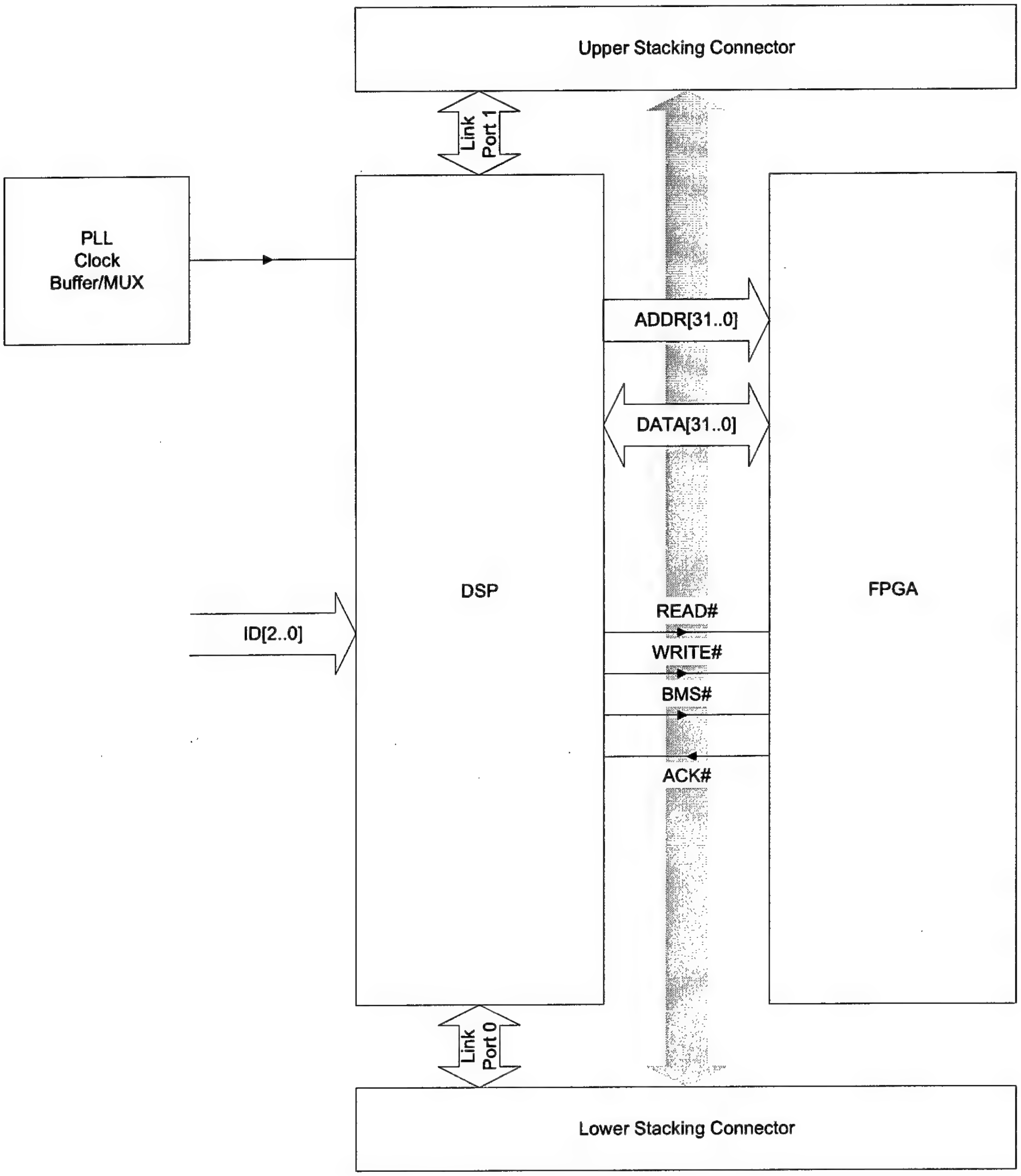


Fig. 2-8 DSP-FPGA Interface

When the DSP sends information to the FPGA, it first sets up the data and addresses, and then controls the READ# and WRITE# signals. The DSP supports different types of peripherals intrinsically such as SDRAM and SBSRAM. To simplify the interface to the DSP, the FPGA interface was chosen to emulate SRAM. While the FPGA was processing the DSP's request, the FPGA would hold the DSP in a wait state until it is finished. The DSP is packaged in a 400-ball Ball Grid Array (BGA) package.

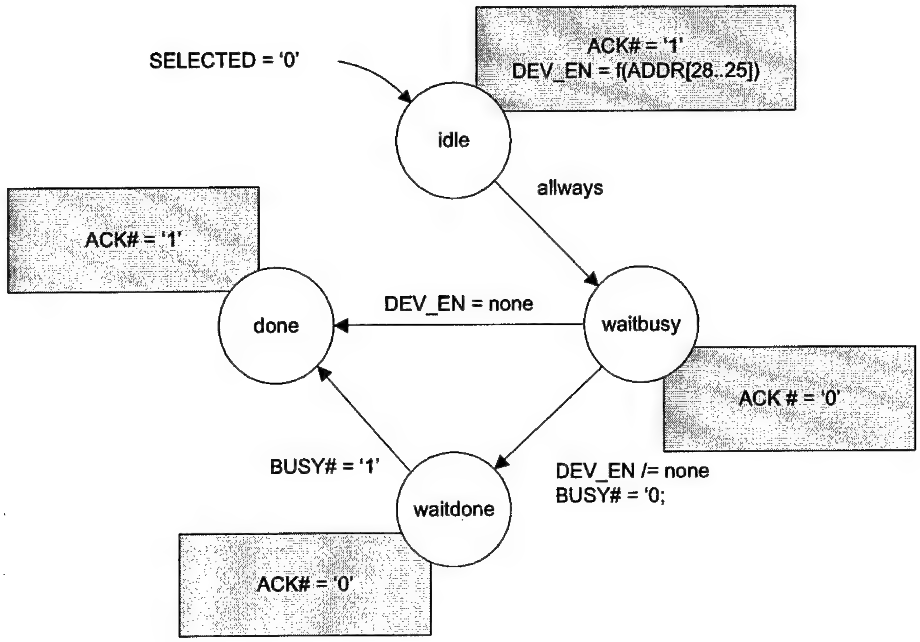


Fig. 2-9 Selector State Machine

The FPGA is selected when ADDR[31..29] is equal to the processor ID as defined by the DSP-ID DIP switches, and when either one of the write signals or one of the read signals are low. Due to the propagation delay within the FPGA, an extra clock cycle was added to the SELECTED signal to allow every signal to stabilize before using those signals to decide on the next state or latch data.

2.3.2 FPGA

On start-up, the FPGA is configured from a Xilinx configuration flash, XC18V04-VQ44C. There are several configuration modes supported by the flash and the FPGA. The one chosen was the parallel SelectMap configuration method. In this method, data is loaded into the FPGA one byte per clock cycle. SelectMap also allows other features for advanced debugging and configuration that can be used via the JM1 and JM2 connectors and the Multilinx programming cable. Due to time constraints, the JTAG interface of the multilinx connector was used.

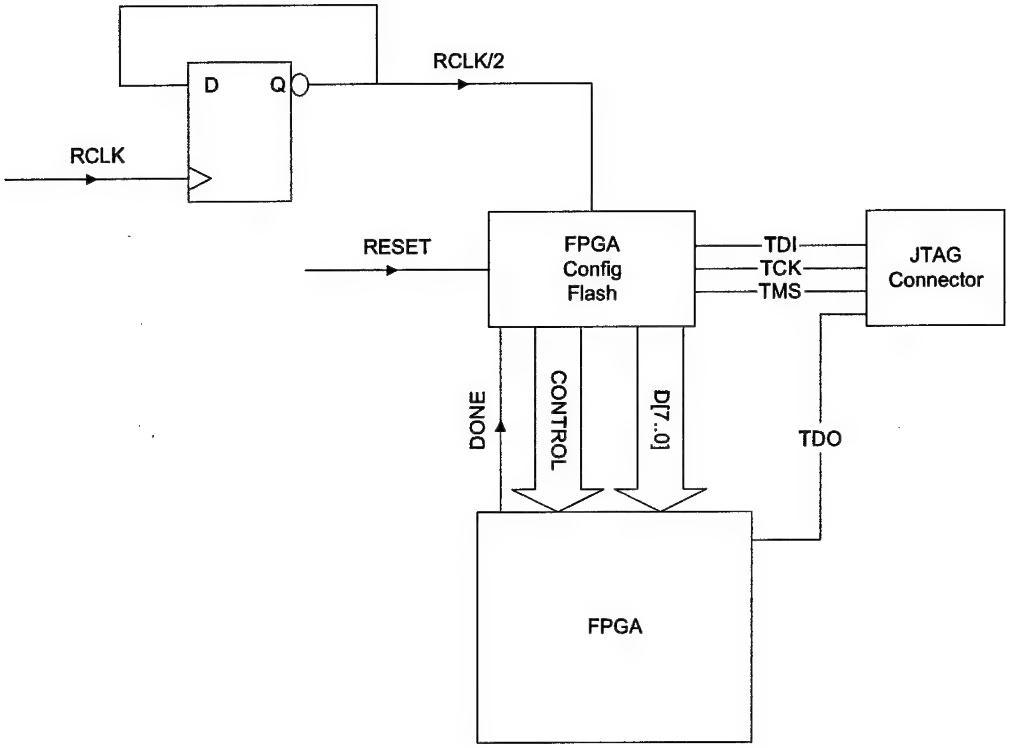


Fig. 2-10 FPGA Configuration Circuit

2.3.3 DAC

There are several different types of DACs, including parallel and serial types. Due to the timing constraints, as well as the availability of a data bus, the parallel type was chosen. This also simplifies the control logic.

The digital to analog converter chosen was an AD8582AR. The AD8582AR is packaged in a 28 pin SOIC package, with 50 mil pitch. This DAC has the following features:

- Two channels
- 12-bit resolution using parallel interface
- 0-4.096 Voltage output range
- 2.5 Volt reference voltage
- 16 microsecond settling time

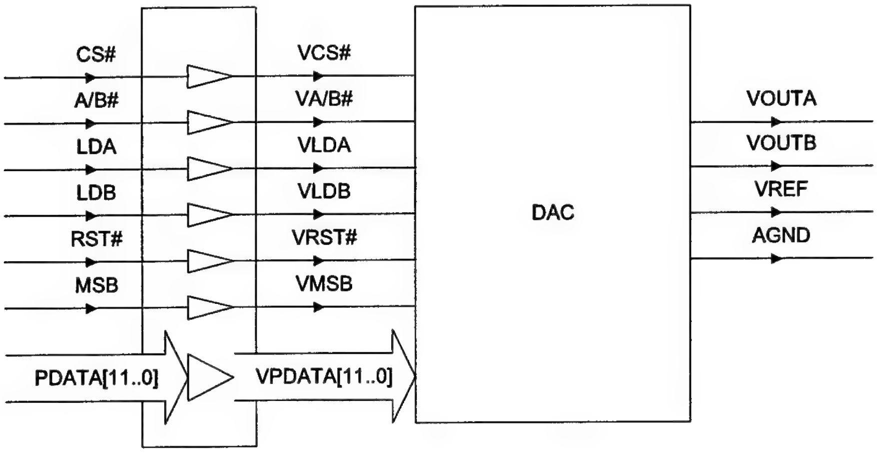


Fig. 2-11 DAC Block Diagram

The DAC is interfaced to the peripheral bus of the controller. The DAC is a 5 Volt device. To make the peripheral bus compatible for devices that are not 5 Volt tolerant, a LVCMOS buffer (Texas Instruments SN74LVC16244-ADGGR) was added between the DAC and the actual bus itself. All signals come from the FPGA. Since the DAC is purely an output device, and sends no control or status signals back to the FPGA, the output enable pins of the LVC16244 buffer could remain active at all times. The chip would ignore the signals until the appropriate control signals are sent.

The control for the DAC consisted of both types of data: that which is sent from the DSP, and lasted the duration of the transaction (referred to here as Transitory Data), and that data which is latched in the FPGA and remains after the transaction (referred to here as Persistent Data). The persistent data for the DAC is the RESET command, RST#, and the MSB signal, which decides during a reset which value the DAC should reset to: 0x000 or 0x800. These two data reside in the control register block. The actual data that changes the value of the analog channels (transitory data) is routed using the DAC block.

The DAC block primarily functions as a timing controller for latching the value into the DAC. The setup and hold times must be valid for the DAC to recognize the new value. The DAC block uses a state machine that allows it to ensure that the data is working correctly.

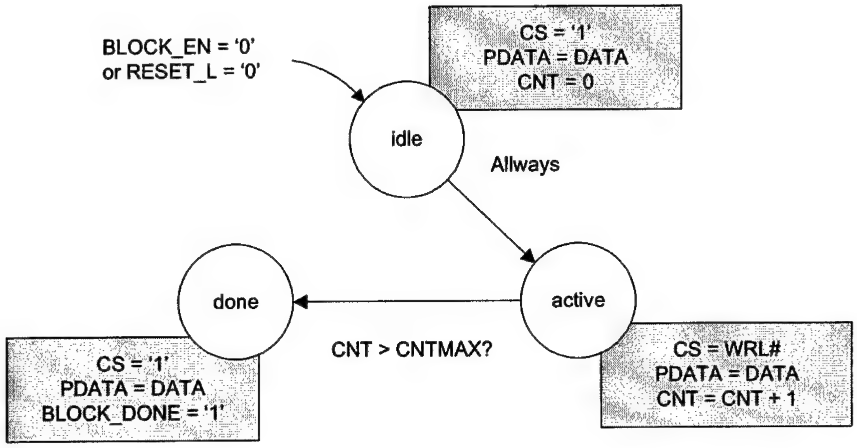
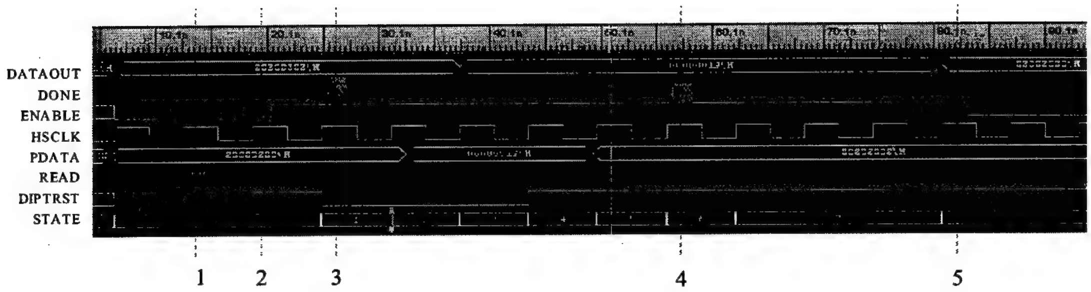


Fig. 2-12 DAC State Machine



TIME	EVENT
1	ADDRESS FROM DSP DECODED AND BLOCK INPUTS SET UP FOR DIP SWITCH
2	SELECTOR ENABLES DIPSWITCH BLOCK
3	DONE GOES LOW, INDICATING BLOCK IS WORKING. DURING THIS TIME, THE BLOCK ENABLES DIP SWITCH OE AND UPDATES BLOCK_DATA_OUT PORT IN FPGA
4	DONE GOES HIGH, INDICATING BLOCK IS FINSIHED, AND DATA IS AVAILABLE OR HAS BEEN PROCESSED
5	DSP DISABLES BLOCK, AND BLOCK TRISTATES

Fig. 2-13 DAC timing diagram

2.3.4 HEX Display

The hex display is used for debugging and visual indication of status. There are two digits displayed. During debugging, these digits can be used to represent a system variable, converter operating state, or setpoint.

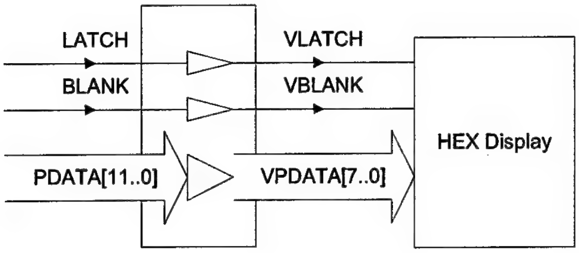


Fig. 2-14 HEX Dispaly Block Diagram

The TIL-311 was chosen because it automatically decodes the four bit data into the correct pattern to represent the corresponding data on the display. No decoding is necessary to implement in VHDL, allowing the data to be directly passed from the DATA bus to the PDATA bus.The hex display is interfaced to the controller via the peripheral data bus. It is a five volt device, and therefore, it exists on the VPDATA bus, which is the five volt extension of the PDATA bus. There are two control lines that are used. The first one is the blank input. When this is high, the hex display does not show any digit, and appears blank. The second control line is the latch input. When this is low, data passes from the VPDATA bus into the hex display. Typically, the data is set up, and then this line is pulsed low to allow the data to propagate into the internal latches in the hex display. When it is high, the hex display ignores the values on the VPDATA bus.

The hex display is a memory mapped peripheral controlled from the FPGA. When the FPGA is addressed, and the selector determines that the address is the address of the hex display, it will set BLOCK_EN to high, and it will then wait for BLOCK_DONE to go low and then high. The following state machine shown in Fig. 2-15 represents the control for the hex display. Data is sent on to the PDATA bus, and it becomes stable while the device is in idle. As soon as the device becomes selected, the state machine moves into the active state. The latch is pulsed low, allowing the data on the PDATA bus to propagate into the hex display. The hex display will remain in this state until the counter expires, indicating that the setup and hold time prescribed in the datasheet has expired. After this, the state machine moves into the done state, setting BLOCK_DONE to high, and setting LATCH back to high, freezing the data. The device returns to the idle state when the selector deselects the hex display control block

The HEX display is in a DIP-14 package. In order to make it surface mounting, a SMT DIP socket was used.

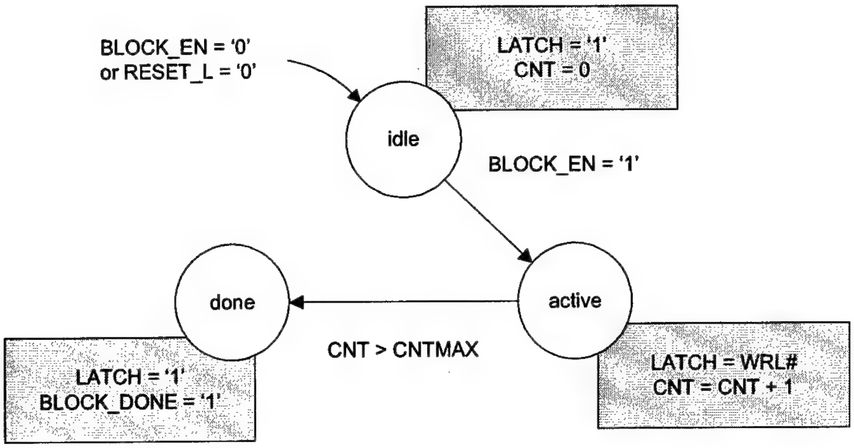


Fig. 2-15 HEX Display State Machine

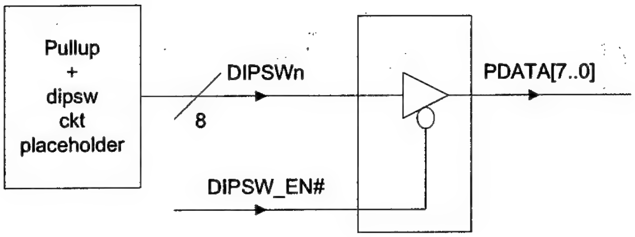


Fig. 2-16 DIP Switch State Machine

2.3.5 DIP Switches

DIP switches are useful for setting parameters that can be used as input to control code. The controller has eight user DIP switches that are general purpose. The DIP switches were interfaced to the peripheral bus due to a shortage of pins. In order to interface them to the bus, a tristate buffer was used. When the PDATA bus is tristated from the FPGA side, the buffer can be enabled to allow the DIP switch data to propagate to the bus. The DIP switch state machine works similar to the hex display, except that it must latch the data once it is available on the hex display. The state machine will not enable the tristate buffers if the DSP does not request a read.

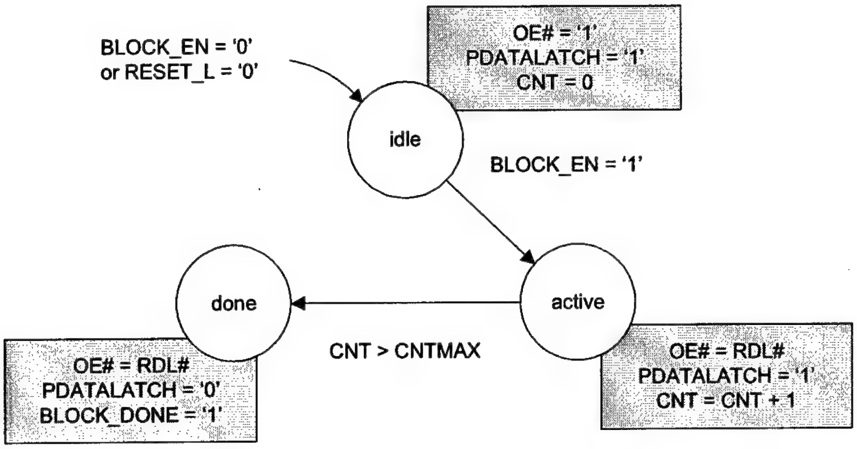


Fig. 2-17 DIP Switch State Machine

2.3.6 PCI Mezzanine Interface

The PCI Mezzanine Card interface is described in IEEE p1386.1, and is a daughter specification to the CMC (Common Mezzanine Card) specification, IEEE p1386. The CMC specification defines the size of a double-wide card to be 149mm x 149mm. The Universal Controller was chosen to be a double-wide mezzanine card because it was impossible to fit all the contents of the controller into a single-wide one.

The PMC card plugs into a host carrier card that is specific to the host type that it is plugging into. For example, in a PC, there are host carrier cards to plug a PMC into the PCI bus. In a VME system, there is a carrier card chipset that can take a PMC, and convert the resulting signals in to ones compatible with the VME system. There also exist similar ones for compact PCI and other architectures. This decision was made so that there was no restriction on the host system type.

The PCI Mezzanine Card Interface can use up to four connectors per slot, referred to as JP1, JP2, JP3, and JP4. JP3 and JP4 are used as I/O connectors. In the Universal Controller, these two are unused. The controller instead uses JP1 and JP2, which carry the PCI signals to the host carrier card.

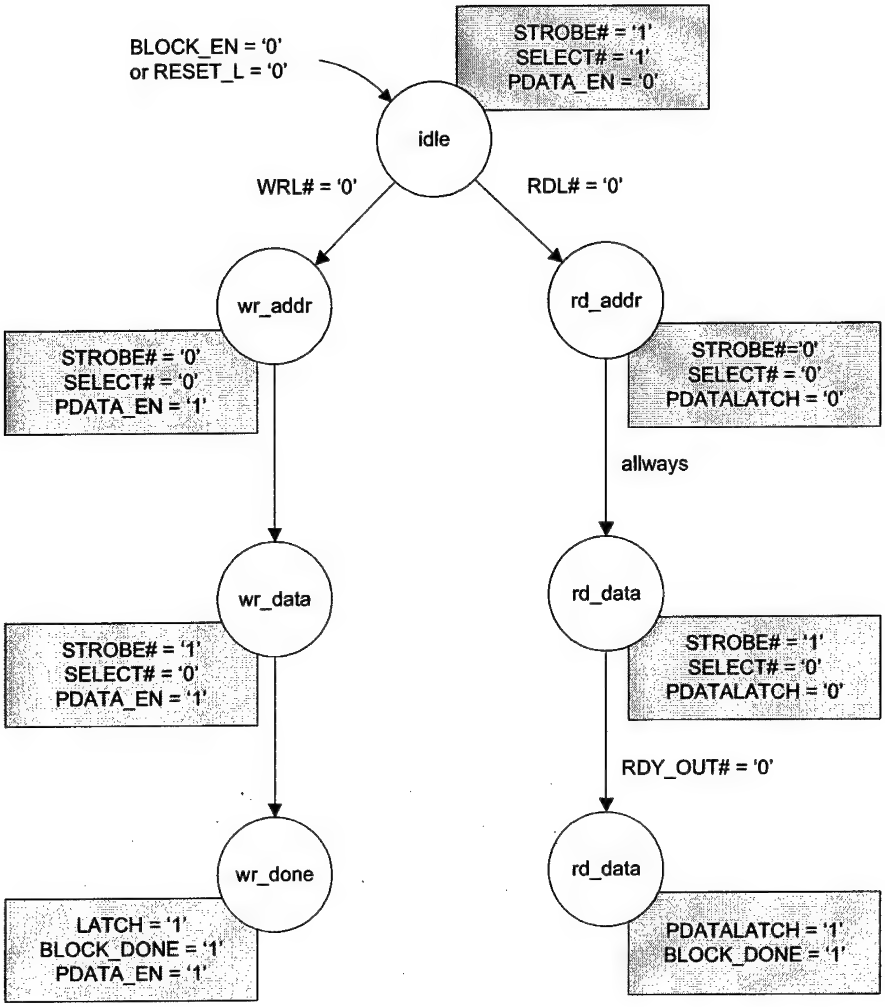


Fig. 2-18 PMC State Machine

2.3.7 DSP Boot Flash

When the controller is powered on or reset, the DSP will use this flash to load instructions and execute control code. While selecting flashes, it is desirable to have one with the following properties:

- Boot Sector
- Sector erasable
- Standardized flash interface

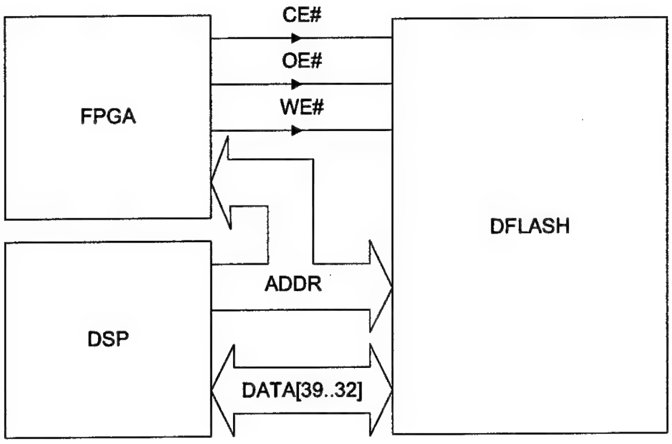


Fig. 2-19 DFLASH – DSP interface.

This flash resides on the DATA bus lines [38..31] as specified by Analog Devices []. There are three control lines going to this device that are the equivalent of a read enable, a write enable, and a chip select. The block diagram of the DFLASH – DSP interface is shown in Fig. 2-19. According to the DSP hardware reference, the DSP addresses the flash via the use of the BMS# pin. This pin is also used to program the flash. However, it is complicated to use this pin, and even the code shipped with the demo board for the ADSP-21160 uses a different method. To take advantage of this already-available code, while still allowing the DSP to boot via BMS#, two methods of activating this flash are implemented. The flash can be activated by addressing a memory location in the range assigned to the flash. This is used to erase and program the flash, as well as to read data from it later in the program. The BMS# pin is asserted by the DSP on startup, and this method is equivalent to selecting the upper address bits to match the range assigned in the FPGA memory map. The demo code shipped with the ADSP-21160 flash, called EZ-KIT [] was modified for the AMD chip. The code was originally written for a ST-Micro flash, which used slightly different commands for writing and erasing.

2.3.8 Peripheral Flash

It is desirable to store data in a flash for diagnosis purposes. After a fault, the flash would provide a time history of the states. Other configuration information could be stored here as well. This flash is similar to the boot flash with the exception that it has no boot sector.

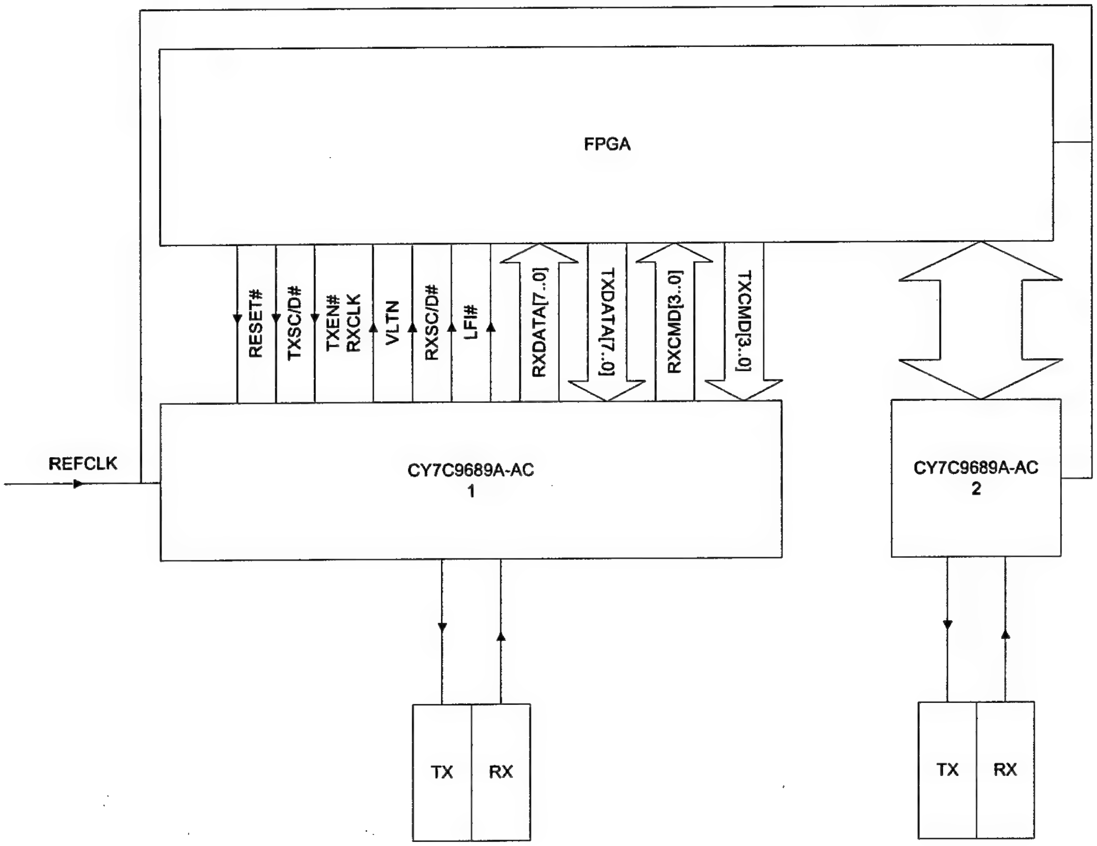


Fig. 2-20 FPGA and Cypress Communications chips interaction.

2.3.9 Fiber Optic Interface

Each fiber optic transceiver is connected to the FPGA directly. An initial version of PESNet has been implemented in the FPGA. However, the second version, PESNET 2.2, remains to be implemented in the future. This protocol supports so far a limited number of nodes.

2.3.10 Peripheral expansion and debug connectors

The peripheral expansion and debugging connectors serve the following functions:

- Debugging VHDL modules
- Interfacing boards that expand the functionality of the controller
- Providing an interface into the system for a logic analyzer
- Bus Monitor

Several examples of expansion would be fiber optic transmitters and receivers used to control peripherals local to the controller, such as crowbars or safety devices. It can also be used to generate PWM signals to control things such as analog and digital meters. They can also be used to implement serial busses to support peripherals such as LCDs.

The peripheral expansion and debug connectors are pins directly connected to the FPGA. These pins can be programmed by writing custom VHDL modules that will control their behavior. These blocks interface to the control register or they receive commands from the data and address lines, enabled by the selector.

One example where this was used was in the verification of the analog to digital converter for the hardware manager. The code was developed and debugged in simulation before the board was ready. Instead of waiting for the real hardware manager to come back, the analog to digital converter control block (written in VHDL) was placed in the controller, and interfaced to the peripheral connectors. It was easy to create an in-house PCB on which the ADC could be placed. Here, the block was verified. When the hardware manager came in, the ADC operation was already guaranteed.

2.3.11 Global control connector

A connector was made available to interface a daughter card that supports a higher level communication protocol for several purposes:

- supervisory control of the converter (controller is a device controlled from a PLC)
- Monitoring of the converter
- Adding distributed periphery to the universal controller (controller is busmaster)

This global control connector is interfaced to the peripheral bus, and supports 3.3V and 5V devices. The 3.3V devices do not have to be 5V tolerant, as the 5V section of the bus is isolated using buffers. Several vendors make chips to support upper level communications. Some of these are listed below.

Table 2-1 List of high-level commercial communication chips

Protocol	Vendor	Chip	Description
Profibus-DP	Siemens	SPC-2	Supports FDL
Profibus-DP	HMI	ABIC	AnyBus-IC single hybrid chip with digital and analog components
Profibus-FMS	Siemens	ASPC-2	
Profibus-DP	Profichip	VPC3+B	Slave ASIC
LonWorks	Cypress	CY7C53150	
DeviceNet	HMI	ABIC	AnyBus-IC single hybrid chip with digital and analog components
ControlNet			
AS-i Bus	ZMD	A2SI-ST	AS-i master
Industrial Ethernet	HMI	ABIC	AnyBus-IC single hybrid chip with digital and analog components

The Global Control Connector has all of the peripheral address and data lines available, as well as the bus control lines. In addition, there are ten lines private lines that are customizable depending on the ASIC that is used on the daughter card. Some of these chips use serial interfaces, in which case two or three of these lines will be used for the interface, and the bus interface is not used at all. The interface code is written in a VHDL module, and placed into the FPGA modular architecture, removing the placeholder for the global control interface.

2.4 Methodology

2.4.1 Universal Controller PCB Design

There are many aspects of implementation that are not considered when designing the logical interconnection of components as in the previous steps. With so many components sharing the same signals, the layout of the controller needs to be considered carefully. The FPGA has 560 connections to the PCB itself within a 3x3cm area. The DSP has 400. Many of these signals are switching at 40 or 80 MHz. It is important to consider the distance that these signals have to travel.

Another consideration is EMI shielding. Several components are noisy, and should be shielded so that the noise does not affect other signals on the PCB. With so many components switching at different frequencies, bypassing and power planes become important to consider. Having planes, in turn leads to issues such as copper balance vertically throughout the PCB.

Another constraint to the PCB design is the mechanical layout specified by the CMC standard (PMC parent specification IEEE 1386). This specification requires the PCB to be 149mm x 149mm with a thickness of 62 mils. There were both good and bad effects of this specification. The good effect is that the signal lengths were reduced, eliminating problems introduced due to long trances. The bad effect is the increased density of signals. The higher density of signals led to blind vias between three layers: Top and Midlayer 1, Top and Midlayer3, and Bottom to Midlayer 6. This added to the warpage of the PCB, and dramatically increased the cost.

Another important area of the PCB was the optical transmitter. This area had very sensitive signals switching at 125 Mbps. This area is sensitive to noise, and so several precautions were taken to ensure its successful operation. First, an exact layout of the transmitter was given by Agilent Technologies, the manufacturer of the optical transmitter and receivers in application note 1066. The GERBER files were obtainable for this circuit. Although it was not possible to directly apply the GERBER files to the circuit, the layout could be copied point by point for every trace to exactly replicate the layout twice in the controller design. Since the design of this transmitter/receiver circuit used 4 layers, and the controller used 8layers, the additional layers were replicated as power planes to increase the noise immunity and add capacitance between the isolated power plane pair.

The final version of the controller had a total of 709 nets, which were connected to a total of 3280 pads. Additional characteristics of the PCB are listed in Table 2-2.

Table 2-2 PCB Attributes

Attribute	Value
Number of Layers	8
Number of Holes	1978
Number of Vias	1912
Number of Components	396
Smallest space between different nets	5 mil
Smallest trace width	5 mil
Mask Type	HASL

While the PCB software, Protel 99SE, had autorouting capability, it was sufficient for such a dense 8-layer PCB. With the help of an external contractor, Gasha Gataric, the first version of the controller was manually routed.

2.4.1.1 Quality Assurance

During the development cycle, there were several points at which the controller was examined by people other than the developer. This is important, as there are implicit assumptions made by the designer that are impossible to recognize by someone who is looking at their own work. To address this, quality teams were assembled to examine the controller for defects in design at several stages. The teams consisted of electrical engineering and computer science B.S., M.S., and Ph.D. candidates. Methods used for quality assurance came from previous work experience in a software development company, as well as from [McConnell Software Project Survival], [Lewis, PMDR], and [Schertz and Whitney, Design tools for Engineering Teams], [Evans and Lindsay, Management and Control of Quality].

QA sessions were held after the completion of the schematics, but prior to the PCB design or modification. Two major sessions were held. The first one was during the design of the first board, and the second one was prior to submitting the modifications for the second board to the PCB manufacturer. Checklists governed the objectives of each QA session. Two teams of two people were assigned to each area of the reviews to avoid group think behavior while at the same time double-checking everything. There were several areas to review during each session.

First, the schematic symbols had to be checked. Each schematic symbol was created from the component datasheet. QA teams checked that each pin of each component matched each name on the schematic symbol. A discrepancy at this point would propagate through the rest of the design, and would be very hard to fix in the future.

The second type of check was to make sure that the pad size and location of each PCB footprint matched the specifications on the datasheets. If the footprint was wrong, then the component would not fit after manufacturing. It is also important to check that Pin 1 was in the same location, as some packages, especially TQFP and PLCC types vary the location of pin 1 from component to component.

The third type of check was a netlist check. It is easy to misspell a net name, or to use different notation in different locations when they should be the same. Examples of this are: +3.3V vs. 3.3V vs 3V3 vs VCC. Similarly, active-low signals have the same problem: BMS, \BMS, nBMS, BMSn, BMS_L. Other issues that can occur in this area are nets that are the same, but they are still separated due to some aspect of the PCB software, Protel 99SE.

For the PCB check, several issues were examined: Firstly, the mechanical dimensions were checked. The CMC specification has several holes for the double-CMC form factor that specify the x-y location as well as the diameter and clearance for each hole. Secondly, aspects of the routing were checked for undesirable features, such as multiple vias in a single trace, as well as traces going past noisy areas of the PCB, such as oscillators and power supplies. Noisy traces such as clocks had guard traces or planes to shield these components. Other sensitive areas were the communication transceivers, which transmitted at 125 Mbps, as well as the transceivers and FPGA. Thirdly, the power planes were checked to ensure that they were connected correctly. In the second PCB check, the copper balance was also checked. A significant defect in the first design was the copper imbalance from top to bottom of the PCB. If the copper is not balanced, the PCB will tend to warp and twist, which in turn makes large surface mount components difficult to solder, especially BGA (ball grid array) devices such as the DSP and FPGA. One PCB failed assembly due to this issue, as the X-RAY revealed bridging between two connections on the corners of the BGAs. This is discussed more in the issues area.

Many of these QA issues resulted from the initial inspection of the PCB and the resulting tests. They were added into the second revision, as the issues were recorded in a database, which in turn created a checklist of issues to explicitly verify in the new design. As issues were found, they were added to a quality database to make a complete history of the issues found in the PCB, who found them, when, what category they were related to, and a history of comments associated with that issue describing what actions were taken to correct it. This database in turn automatically produced a release checklist that was reviewed to ensure that all issues were accounted for before manufacturing the second design.

During several points in the design and development cycle, the design was presented to project stakeholders for review and comments. Using their feedback, the design was modified to include their input. Outcomes such as the use of PMC resulted from these discussions. In addition to these changes, one stakeholder (General Dynamics Advanced Information Systems) conducted another PCB design review, and feedback to each change was discussed ring weekly teleconferences that lasted from 30 minutes to 90 minutes. The first revision of the PCB turned out very successful. There were some issues with equipment and software, which took time to fix, but once these issues were solved, there were not a lot of problems left, as the QA sessions identified many problems that would have rendered the PCB unusable if they were not identified.

2.4.2 Issue Tracking Database

It is important in complex designs to keep track of problems so that they may be fixed before the next release. A database was designed that would maintain a list of issues, as well the history related to that issue. This database facilitated the QA process. The fields considered are shown in Table 2-3.

Table 2-3 Issue Tracking Database

Field Name	Description
IssueNumber	A unique id that describes the issue
Issue Title	A brief description of the issue
Date Opened	The date that the issue was found
Date Closed	The data that the issue was closed
Assigned To	The person who is responsible for resolving the issue
Assigned By	The person who submitted the issue
Issue Type	**Issue:** **Assignment:**
Issue Status	**Open:** Issue is submitted. **InProgress:** Issue is being worked on **Resolved:** Issue has been fixed **Closed:** Issue has been verified and closed. **Reopened:** Issue has been fixed, and verified, but now it is again a problem, or a canceled issue has become of interest again **Canceled:** Issue will be ignored
Issue Severity	**Critical:** Board cannot be manufactured with this defect, or it will be unusable **High:** Board can be manufactured with this, but the functionality will be severely impaired **Medium:** Board can be manufactured with this, but some features will be disabled **Low:** Board can be manufactured, but some features will not work exactly as designed **Cosmetic:** Board functionality will not be affected
Issue Priority	**High:** Issue must be resolved ASAP, because it will impede future work on the controller until it is addressed. **Medium:** Issue may have a complicated fix that should imply that it should be fixed before it becomes harder to fix as more work is done. **Low:** Issue is not of immediate concern.
Project	A description of which project this issue belongs to
Subproject	A description of which part of the project this issue belongs to
Description	A more detailed description of the problem or how to reproduce it
Comments	As work is done, this field is updated to keep track of changes, status, and concerns associated with this issue. Each entry automatically places the date and submitter of the comment in the field.

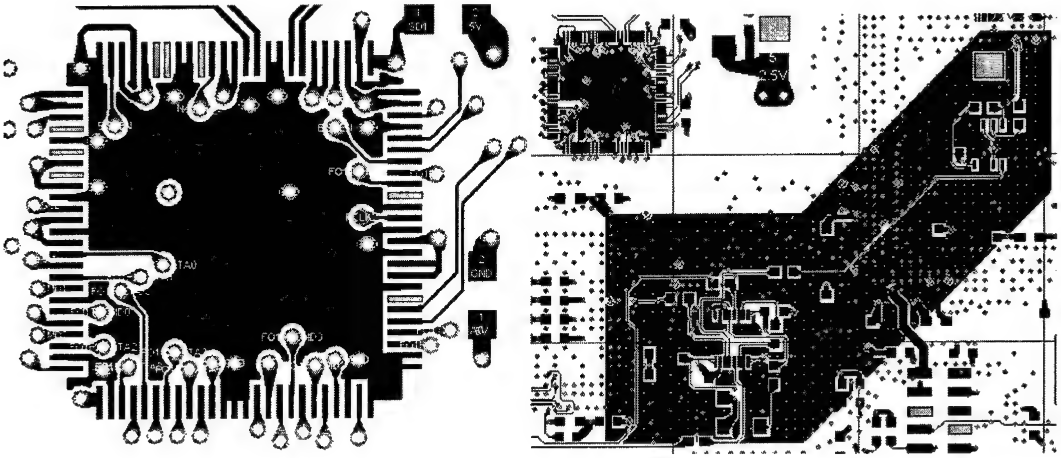


Fig. 2-21 Shielding Planes

2.4.3 Revisions and Final Design

After receiving the first design, feedback from the assembly house on the fabrication house, feedback from General Dynamics, and in-house testing, several modifications were planned for the second version of the PCB. One of the significant improvements was the power plane bypassing capacitors. Originally, the capacitors had traces that made a circular path from the capacitor to the via. In the second design, the vias were moved much closer to the bypassing capacitor pads, and larger traces were used to connect to these. General Dynamics also suggested that the bypass capacitors should not be connected to the pins of the device. Instead, the purpose of the capacitors should be to bypass the plane beneath the device, and other vias should bring that power up to the component as close as possible. Several changes were made, especially next to the Cypress transceivers to accommodate these suggestions.

The issue of copper balance significantly affected the PCB. To address this, additional copper, connected to the ground planes, was added to the perimeter of the PCB in the unused areas of the midlayers. Additional, larger planes were added to the top and bottom of the PCB, and were able to add shielding to clock buffers, oscillators, and the communication chips, as shown in Fig. 2-21. Based on feedback from the assembly house, the finish was changed from LPI to HASL, as it was easier to mount the BGA components. Other issues identified by the manufacturer include a lack of solder mask on some the top of some vias. This becomes a problem with BGA components, as it allows the solder ball on the BGA to migrate into the via hole, creating an open circuit between the device and the pad.

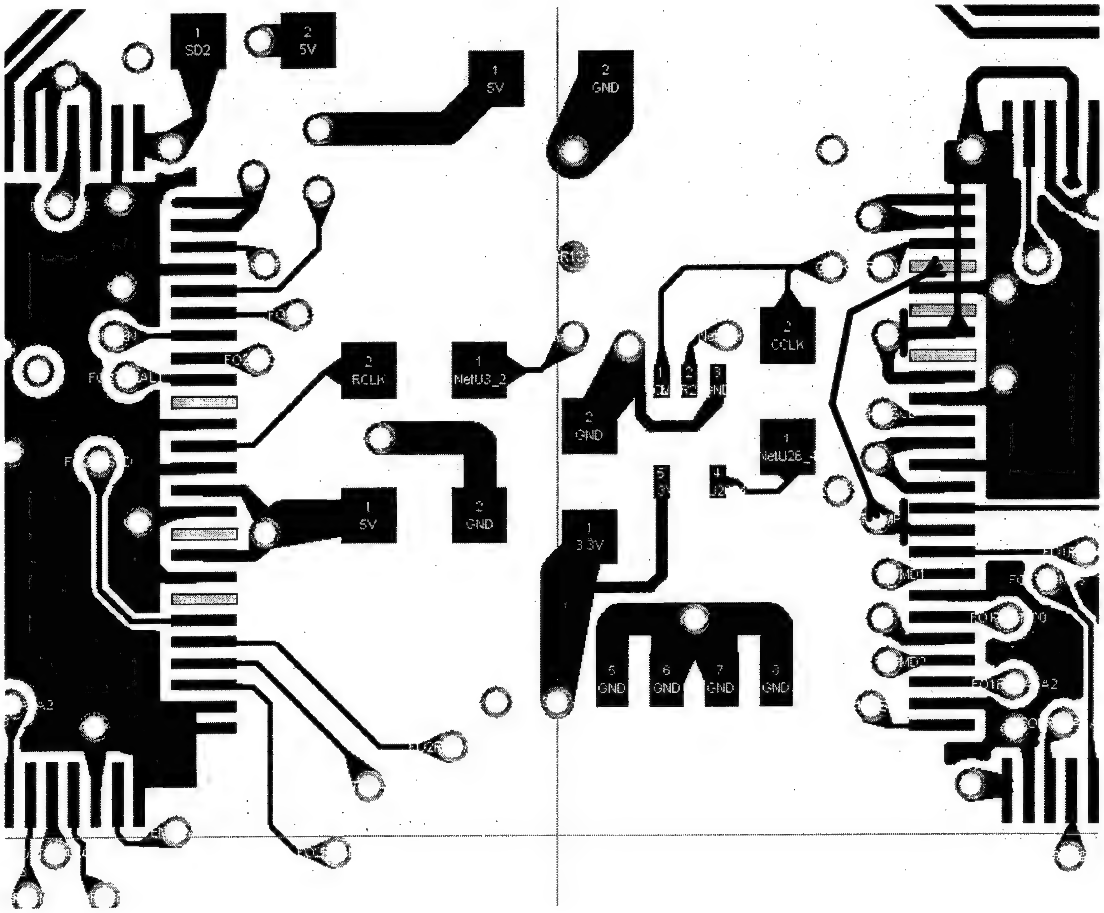


Fig. 2-22 Modifications to U24

2.4.4 Final Design Modifications

After receiving the controller back, most functionality was verified. The fiber optic components required additional modification in order to ensure that they were functional. Two cuts are required on U24, and two jumpers need to be created, as illustrated in Fig. 2-22. The two black lines indicate the jumpers, and the two red lines indicate the cuts. Note that this image is mirrored when looking at this side of the board. That is, U24 is to the left of U25 when the communication chips are facing up.

3 SOFTWARE ARCHITECTURE

3.1 Introduction

A power electronics control system is a real-time system—it operates using limited resources and under a set of deadlines. As distributed control architectures become commonplace for power electronics systems, the size and complexity of the corresponding control software will increase. As a result, the design and specification of the overall software structure become more significant issues than the choice of algorithms and data structures used in the computation [i]. Structural issues in software design include the organization of a system as a composition of components; global control structures; protocols for communication, synchronization, and data access; allocation of functionality to design elements; composition of design elements; physical distribution; scaling and performance; and selection among design alternatives. This is the architectural level of software design.

The traditional procedural or imperative approach to designing embedded control software results in a main-program-and-subroutine architecture that has several disadvantages. The control software is hard to maintain and modify. The software is tightly coupled to the hardware. New systems typically require significant redesign effort, because the main-program-and-subroutine architecture does not support software reusability well.

To address these shortcomings, we present a different approach to structuring software designs. Dataflow is a style of software architecture that strongly supports reusability and reconfigurability [ii]. In the dataflow style, a control application is implemented as a set of concurrently executing processes, which we call elementary control objects (ECOs). ECOs communicate through one-way message queues called data channels. Each ECO is independent, and knows nothing about the other ECOs in the application—it merely consumes data from some channels and produces results on other channels. Dataflow computing is reminiscent of signal filtering and processing, and leads one to design ECOs that are modular and reusable. Constructing control applications then becomes the process of picking ECOs from a library and "plugging them together" into the desired pattern.

This chapter will be organized as following. Section 3.2 gives an overall description of dataflow architecture. Section 3.3 presents dataflow implementation of quite a number of applications. Sections 3.4 and 3.5 present the real-time kernel design in C and C++ respectively. In section 3.6 the communication protocols of transparent messaging between multiple dataflow software is described. And finally section 3.7 compares the dataflow approach with a widely commercially used software development platform.

3.2 Dataflow Architecture

We propose an alternative software architecture for developing control software by composing reusable modules: dataflow [iii], [iv], [v]. This architectural style has the following properties:

- It minimizes coupling between software components.
- It encapsulates hardware dependencies.
- It encourages highly reusable components.
- It supports component-level and architecture-level reconfigurability.
- It allows transparent, distributed communication between components, and thus supports distributed execution naturally.
- It addresses issues of scalability, expandability, and upgradeability.

3.2.1 An Overview of the Dataflow Architectural Style

Software written using a dataflow architecture consists of a collection of independent components running in parallel that communicate via data channels; such a design can be succinctly depicted graphically, as shown in Fig. 3-1. In Fig. 3-1, each node is a computational component and each arrow is a buffered data channel. Each concurrently executing node is a self-contained software part with well-defined behavior. Data channels provide the sole mechanism by which nodes can interact and communicate with each other, ensuring minimal coupling and greater reusability. Data channels can also be implemented transparently between processors to carry messages between components that are physically distributed. Choosing this component model for embedded control software alleviates many of the negative aspects of the more traditional main-program-and-subroutine organization. More importantly, however, it also opens up the possibility of developing a library of commonly recurring, standardized control software functions encapsulated in reusable dataflow components. Such a design library would allow a new control application to be configured rapidly from an existing collection of components.

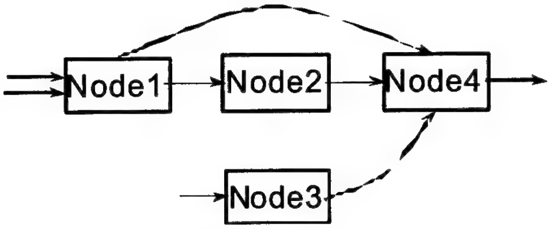


Fig. 3-1 Dataflow architecture.

3.2.2 Elementary Control Objects (ECOs)

In this report, dataflow nodes are called elementary control objects (ECOs) to reflect their role as the building blocks of larger control applications. Each ECO manipulates the input data that it receives according to its behavior, generating output that can be connected to other ECOs. There are no explicit calls between ECOs—in fact, an ECO has no knowledge of the other nodes the system comprises, or of the identities of the other nodes with which it communicates. This inherent independence allows ECOs to be treated naturally as concurrently executing, active objects. This natural concurrency, together with the ability to transparently map data channels across physical component boundaries, provides a direct and simple mechanism for supporting the distributed control of power electronics systems.

An ECO contains an input and output data channel description, a startup parameter description, and an implementation, as illustrated in Fig. 3-2. Three distinct types of ECOs exist within the embedded control domain—computational ECOs, coordination ECOs, and driver ECOs. A computational ECO embodies some specific computational behavior needed for an application. Fig. 3-3 shows several examples of computational ECOs. A coordination ECO, on the other hand, is designed to support transparent management and control of distributed system hardware assets, as exemplified in Fig. 3-4. Driver ECOs encapsulate hardware dependencies and provide a standard program interface to control hardware. Fig. 3-5 shows an A/D driver ECO.

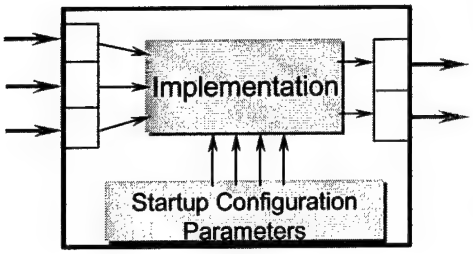


Fig. 3-2 ECO structure.

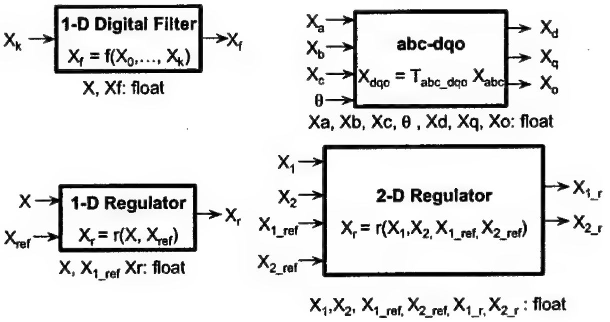

1-D Digital Filter
$X_f = f(X_0, \dots, X_k)$
X_k
X_f
X, Xf: float
abc-dqo
$X_{dqo} = T_{abc\_dqo} X_{abc}$
X_a
X_b
X_c
θ
X_d
X_q
X_o
Xa, Xb, Xc, θ , Xd, Xq, Xo: float
1-D Regulator
$X_r = r(X, X_{ref})$
X
X_{ref}
X_r
X, X_{1_ref} Xr: float
2-D Regulator
$X_r = r(X_1, X_2, X_{1\_ref}, X_{2\_ref})$
X_1
X_2
X_{1_ref}
X_{2_ref}
X_{1_r}
X_{2_r}
$X_1, X_2, X_{1\_ref}, X_{2\_ref}, X_{1\_r}, X_{2\_r}$: float

Fig. 3-3 Examples of computational ECO.

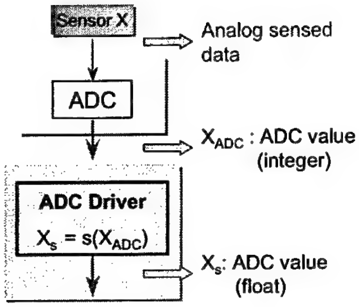


Fig. 3-4 Example of a coordination ECO.

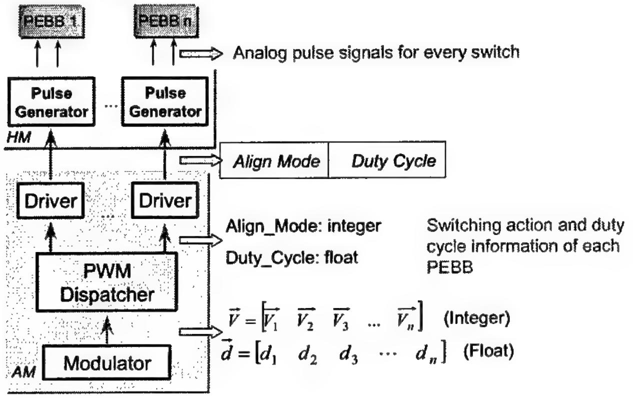

PEBB 1
PEBB n
Analog pulse signals for every switch
Pulse Generator
Pulse Generator
HM
Align Mode
Duty Cycle
Driver
Driver
Align_Mode: integer
Duty_Cycle: float
Switching action and duty cycle information of each PEBB
PWM Dispatcher
$\vec{V} = [\vec{V_1} \ \vec{V_2} \ \vec{V_3} \ \dots \ \vec{V_n}]$ (Integer)
$\vec{d} = [d_1 \ d_2 \ d_3 \ \cdots \ d_n]$ (Float)
AM
Modulator

Fig. 3-5 A/D driver ECO.

3.2.3 Data Channels

Data channels serve as the sole communication paths connecting ECOs into a cohesive control algorithm. Each data channel connects a pair of ECOs: the source ECO generates data and the sink ECO consumes data. Note that data channels are unidirectional—data can only flow from one source ECO to one sink ECO. Data channels carry typed data based on the application requirements; strong typing helps detect certain kinds of interconnection errors early during development, rather than later during operational testing. Each data channel has a data queue to buffer data between ECOs operating at different speeds. The application designer configures each data channel's data type, buffer size, source connection, and sink connection are when developing the overall software structure.

3.2.4 Dataflow Graph

The dataflow graph describes the control software configuration as a composition of ECOs interconnected with data channels. Fig. 3-7 shows the dataflow graph of control for a close-loop 3-phase inverter. Annotations on the graph specify ECO startup parameters, ECO priorities, ECO execution policies, data channel property choices, and data channel buffering policies. Designing a control application involves constructing such a dataflow graph by selecting ECOs from the design library and connecting them together. Additional user-defined or application-specific ECOs are also easily supported. ECOs within the dataflow graph can be allocated to different processors for distributed execution.

3.2.5 Dataflow Architecture Real-time Kernel (DARK)

Unlike applications that are built up from simple subroutines, dataflow applications require support for their unique features, including support for concurrent execution, data channel buffering, interprocess synchronization, and interrupt handling. One approach to providing this infrastructure is to encapsulate it in a small, embeddable, real-time operating system (RTOS). Such RTOSes are often called “micro-kernels” because, in comparison to full-featured RTOSes, they are stripped of all but the most minimal features to provide extremely efficient services in a minimal memory footprint.

3.3 Dataflow applications implementation

We implemented the embedded control software for quite number of applications using dataflow architecture. The open-loop 3-phase inverter is the simplest application, while the 4-leg inverter is a fairly complicated application.

3.3.1 Open-loop 3-phase Inverter

The DFG for the open-loop 3-phase inverter application is shown in Fig. 3-6. This is the simplest application that we have used in our experiments. The control algorithm is sinusoidal PWM (SPWM). It consists of three *Lookup_Sin* ECOs that receive a *Start* signal from their boolean input data channels. They look up a value from a circular table that they maintain using a table pointer. After every look-up, the table pointer is incremented. The table source and the modification step for the table pointer are stored as part of the ECO’s configuration information. The output values of the three ECOs have a phase difference of 120 degrees. These in combination, form input to the *Modulator* and fire it to produce three floating-point results that

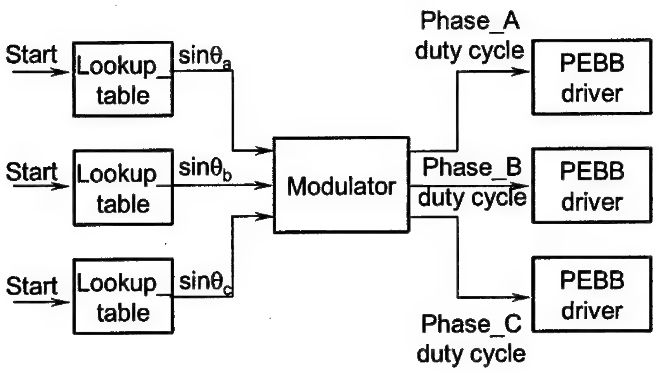


Fig. 3-6 Open-loop three-phase inverter.

form inputs to the three *PEBB_drivers*. The *PEBB_drivers* convert the data from floating-point format to a format of control information that can be understood by the power stage, which they will be inputs to the following two applications are relatively more sophisticated applications than the open-loop application.

3.3.2 Closed-loop 3-phase Inverter

A switching cycle in the closed-loop 3-phase inverter shown in Fig. 3-7 starts with the firing of the two *Lookup_Sin* ECOs by external interrupts, and the input of external feedbacks from A/D converters to the *Abc_Dqo* ECO.

The outputs of the *Lookup_Sin* ECOs go to 1-2 duplicators because the same output has to be duplicated for both the *Abc_Dqo* and the *Modulator*. On receiving both the feedback information and the outputs from the *Lookup_Sin* ECOs, the *Abc_Dqo* ECO transforms the input abc coordinates to dqo coordinates. These dqo coordinates form inputs to the 2-d regulator, which performs PI regulation to get duty cycles in the dqo coordinates. The *dqo_albe* ECO transforms the dqo coordinates back to αβγ. After this, the 3-d modulator performs modulation on the duty cycles. The generated duty cycle information forms input to the power stage.

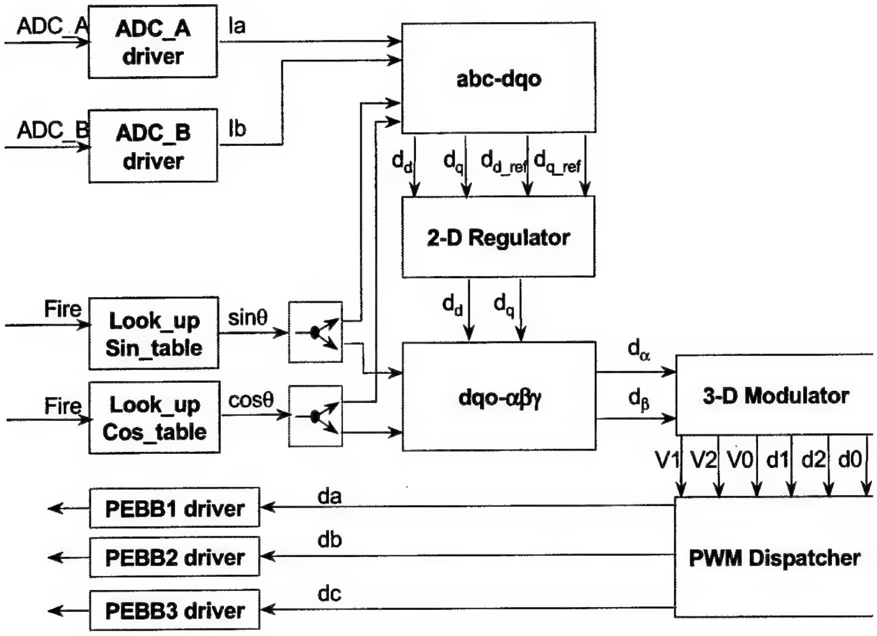


Fig. 3-7 Closed-loop three-phase inverter.

3.3.3 Boost Rectifier

Fig. 3-8 shows a dataflow graph for a 3-phase boost rectifier closed loop control. There are two control loops: current loop and voltage loop. The voltage loop should be executed first to generate reference for the current loop. For the 3-phase current loop control, the dq transformation technique is used.

All sensed data are implemented as interrupt-driven data channels, synchronized by the switching clock. The rising edge of the switching clock causes the execution of the corresponding interrupt handler, in which all the sensed data, phase currents and voltages and dc voltage are updated. Those ADC drivers then translate those sensed data to correct values, respectively. At this point ECO *1-D regulator* and *synchronize* are ready. The dc voltage is regulated in the ECO *1-D regulator*, and the current loop reference d_{d_ref} is generated. The *synchronize* ECO tested the phase voltages and generate a boolean output to indicate whether the phase angles need synchronizing.

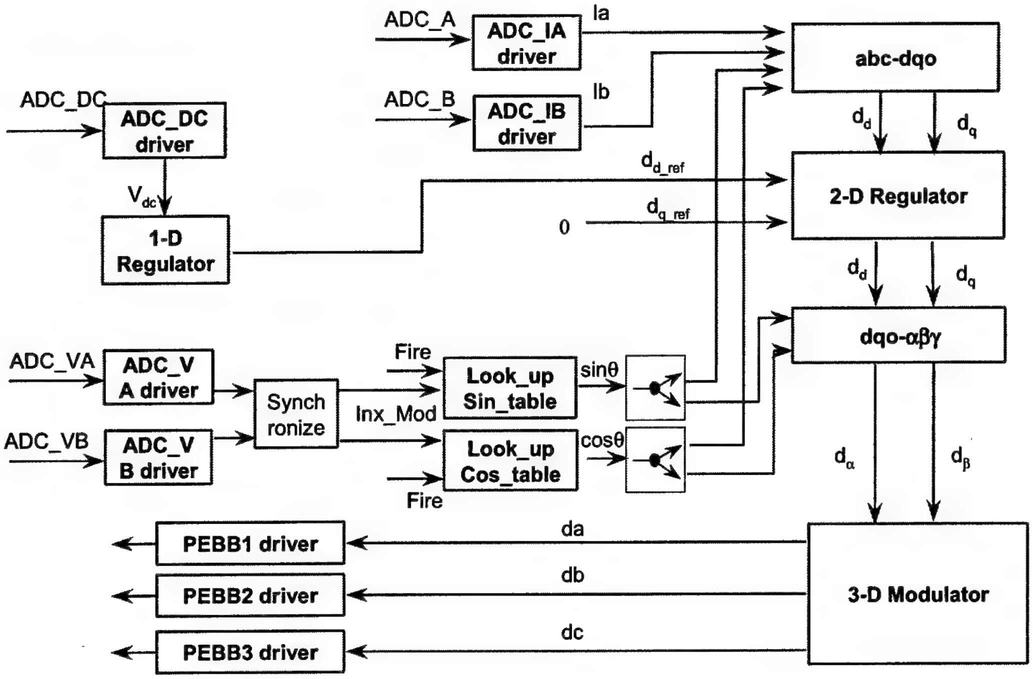


Fig. 3-8 Closed loop control for 3-phase boost rectifier.

The two *lookup_table* ECOs have two different behaviors depending on different inputs combinations. At normal condition, if the phase voltages do not need synchronizing, the *lookup_table* ECOs increment their table pointers and output the table value; otherwise, the table pointers will be reset. Through ECO *duplicator*, the sin and cos values are copied and directed to two different ECOs. So far, the *abc_dqo* ECO is ready to transform the phase currents in abc coordinates to dqo coordinates. Then the *2-D regulator* performs the current loop regulation with the reference generated from the voltage loop. The regulated currents in dqo coordinates will then be transformed back in αβγ coordinates through ECO *dqo-αβγ*. Then the ECO *3-D Modulator* is ready to synthesize duty cycle information for each phase. In the *PEBB_driver* ECO, the duty cycle information will be translated to generate switch pulses at the phase leg gates.

3.3.4 Closed-loop 4-leg Inverter

Fig. 3-9 shows the dataflow graph of control for a closed-loop 4-leg inverter. Dq transformation technology and SVM are still employed in the application. Since this application contains four phases, so the dq transformations and SVM has one more dimensin than those are used in the previous applications. And the regulation of output voltages is decoupled into three independent regulations.

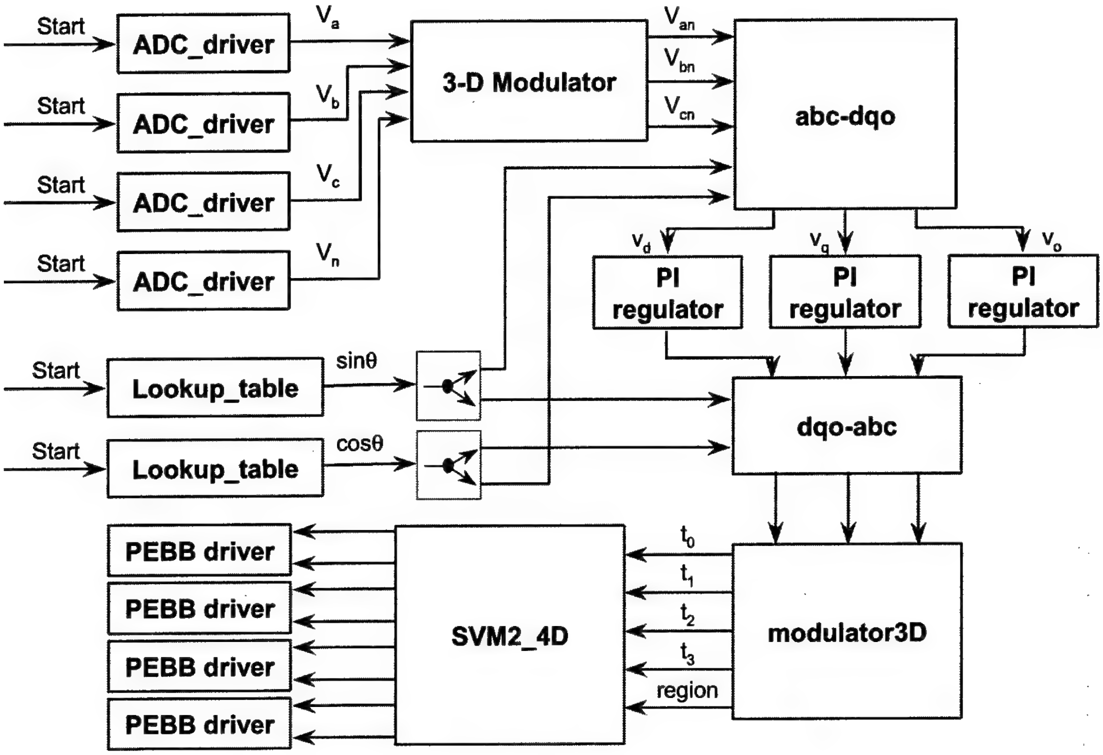


Fig. 3-9 Dataflow graph of control for closed-loop 4-leg inverter.

3.4 Dataflow Architecture Real-time Kernel (DARK)

This section provides an insight into the architecture of **D**ataflow **A**rchitecture **R**eal-time **K**ernel (DARK). Firstly, the requirements of the dataflow applications imposed on DARK and specific demands of power electronics control applications is discussed. Then the kernel infrastructure is presented, followed by a discussion of some prominent kernel features, higelightened by a description of the configurable options of DARK. Finally, the DARK design is assessed experimentally.

3.4.1 Real-time Kernel Design Requirements

As presented in section 3.2, dataflow software is structured significantly different from the legacy control software, which imposes different requirements on the underlying platform or the kernel [vi].

The main requirements imposed on the underlying kernel by dataflow applications are listed below:

1. ***High Performance***: The components of dataflow are used to replace the equivalent hardware components in power electronics controllers. In these cases, the execution speed of the control software becomes an important factor because software is generally slower than the hardware. So, the kernel for these applications should have minimal overhead and high execution speed.
2. ***Faster Context Switching***: Dataflow applications tend to have a larger number of processes or threads than applications developed using other techniques. While a larger number of independent, reusable, components makes application design easier, it can lead to an increase in the amount of context switching overhead. The kernel should make an attempt to reduce this overhead by increasing the speed of context switches as well as limiting the number of context switches.
3. ***Efficient Inter-Component Communication***: The fact that processes only communicate via data channels implies that there are frequent (but usually small) interactions between processes along these channels. Ensuring mutually exclusive access to critical state within a data channel provides another potential for increased overhead in dataflow applications. The kernel should provide support for efficient inter-component communication with minimum overhead.
4. ***Dataflow Scheduling***: Unlike traditional processes that are scheduled based on their priorities alone, dataflow processes are scheduled on the basis of both the priorities and data in the incoming data channels. Moreover, the dataflow processes should not be awakened by every incoming message. The kernel should provide an efficient mechanism to specify when a dataflow process is ready to execute.
5. ***Component Execution with Dynamic Priorities***: A dataflow process can wake up due to data in different sets of incoming channels. Depending on the set, it can take specific actions. The kernel should facilitate this, in addition to adjusting the process priorities according to the actions they are taking.

3.4.2 DARK Architecture

DARK is implemented in C, with a few key elements in assembly (context switching, dual register set support, and interrupt handling). Because it is intended for embedded power electronics control, it currently runs on Analog Devices SHARC 21xxx 32-bit digital signal processors. Dataflow processes, or ECOs, are implemented as C functions. DARK uses a statically initialized array of ECO descriptors, together with a statically initialized array of data channel descriptors, to initialize the application at startup.

3.4.2.1 Kernel Components

As discussed in 3.2, dataflow applications are composed of two building blocks: nodes and data-channels. DARK uses threads to encapsulate dataflow nodes or ECOs, whereas the data-channels are implemented as circular message buffers or mailboxes.

3.4.2.1.1 Threads

For concurrent execution, ECOs could be implemented as either separate processes or separate threads. Processes run in separate address spaces and include program code and current activity, as represented by the value of program counter and the contents of the processor's registers. A process also contains a runtime stack, containing temporary data (such as subroutine parameters, return addresses, and temporary variables), and a data section containing global variables. A thread, on the other hand, is an entity capable of executing concurrently with other threads and has its own runtime stack. Unlike processes, threads run together in a single address space. The threads share with peer threads their code section, data section, and operating system resources such as open files and signals. The extensive sharing makes CPU switching among peer threads and the creation of threads less expensive, compared with context switches among heavyweight processes. Although a thread context switch still requires a register set switch, no memory-management-related work needs to be done. On the negative side, like any parallel processing environment, multithreading may introduce concurrency control problems that require the use of critical sections or locks.

Threads incur less overhead, allow faster interprocess communication, are more memory-efficient, and support faster context switching. These are the reasons why threads are preferred over processes in embedded systems, despite their inability to provide memory protection. As a result, DARK maps each ECO or node in the dataflow graph (DFG) to a separate thread (thread and ECO are used interchangeably hereafter).

For each thread, the user provides the stack size, priority, ECO function, and firing rule. The stack size required for a thread depends on the number of local variables used and the nesting of function calls in the ECO's code. The priority given by the user is the initial priority assigned to the thread when the application starts. The kernel uses the firing rule associated with a thread for scheduling. All of the runtime information associated with an ECO is stored in a structure called Thread Control Block (TCB). There is one TCB associated with each ECO or thread. The TCB serves as the repository for any information that may vary from thread to thread. The structure of a TCB in DARK is given in Fig. 3-10.

```
typedef struct
{
   ECO_Data                    p;
   ECO              eco;
   Context          thread_env;
   Thread_State     thread_state;
   Firing_Rule      firing_rule;
   Firing_Mask      in_ports_ready;
   int              wakeup_time;
   int              deadline;
   unsigned int*    stack_pointer;
   unsigned int     stack_size;
   bool             in_OS_call;
} TCB;
```

Fig. 3-10 Thread control block.

All of the information in the TCB is initialized during application startup. The ECO_Data structure p is the static descriptor used to initialize the ECO. It contains information provided by the application developer that is used by the ECO during its execution. Each ECO is implemented as a C function. The eco field points to the function that represents the ECO. The Context structure is used for saving and restoring the runtime environment of the thread during context switching. The current thread state is stored in thread_state. The firing_rule and in_ports_ready fields are used for scheduling, whereas wake_up_time is used for time management. The variable deadline stores the time by which the ECO should finish execution to meet the real-time deadlines of the application). There is a separate runtime stack for each thread. The field stack_pointer points to the stack space allocated for the thread. The size of the stack is given by stack_size. The field in_OS_call is used for synchronization of threads to prevent shared data problems.

3.4.2.1.2 Data-Channels

DARK maps each arc in the DFG to a typed data channel that is implemented as a circular byte buffer. Each data channel has a Queue Control Block (QCB) that stores the information shown in Fig. 3-11.

```
typedef struct
{
   int                  DC_id;
   Type_Tag                       type;
   short int            element_size;
   Array_Descriptor array_dimensions;
   Overflow_Style       overflow_style;
   int                  front;
   int                  rear;
   int                  size_in_bytes;
   volatile int         size_in_elts;
   volatile int         num_entries;
   bool                 blocked;
   bool                 interrupt_driven;
   Process*             source_thread;
   Process*             sink_thread;
   char                 buffer[1];
} QCB;
```

Fig. 3-11 Queue control block.

Each data channel has an id used for internal management. The `type` field stores the type of elements that can be stored in the data channel. This type tag can be used for run-time type checking through the DARK API in debug builds; alternatively, this feature can be turned off using preprocessor definitions to eliminate the corresponding overhead. DARK data channels support all primitive data types along with complex data types like multi-dimensional arrays, strings, and uninterpreted byte vectors. The field `element_size` stores the size of one element in the data channel, whereas `array_dimensions` stores the array dimensions, if the data channel is of array type. The application designer must specify what happens when a data channel's source ECO attempts to write new data while the channel is full. The writing ECO may block until space is available, overwrite the newest element, or overwrite the oldest element. The `overflow_style` field determines the action to be taken in such cases. The variables `front`, `rear`, `size_in_bytes` and `size_in_elts` are used for queue management. The `num_entries` field contains the current number of elements in the queue. The status variable `blocked` is true when a thread is blocked while attempting to read from or write to the data channel. The data channels can have two entities as their source nodes: ECOs and Interrupt handlers. If the source for a data channel is an interrupt handler, then it is called an interrupt-driven data channel. The field `interrupt_driven` is true for such channels. The variables `source_thread` and `sink_thread` point to the source and sink ECOs respectively.

The QCB block together with the space for the data channel's element buffer is allocated as a single contiguous chunk of memory. This technique allows the element storage space to be accessed as an array by using the `buffer` field. Further, the kernel allocates all QCB blocks and element storage segments in one large block to reduce dynamic memory management overhead.

DARK provides a simple API to the user for interacting with data channels. The user reads from or writes to a data-channel by calling a function of the form: `<operation>_<type>_DC()`. Here the *operation* is either "read" or "write," and the *type* is the type of data to be read or written. Separate functions for each data-type aid in type checking. Internally, these functions call a single OS operation to read or write raw bytes. Other functions for obtaining the status of a data channel or flushing all entries are also provided. The API will be discussed in detail in the next chapter.

3.4.2.2 Kernel Features

This subsection provides an in-depth explanation of the important features of DARK. We start by discussing the Scheduling approach taken by DARK, designed especially for dataflow applications. First we describe the different states of the threads. The high speed context switching provided by DARK is discussed next. Approaches taken for the time management and interrupt handling are described. Then we explain a simple approach taken by the kernel for preventing shared data problems. The limited real-time support provided by the kernel to monitor deadlines is also discussed.

3.4.2.2.1 Scheduling

When more than one thread or process is runnable, the operating system must decide which one to run first. The part of the operating system concerned with this decision is called the scheduler, and the algorithm it uses is called the scheduling algorithm.

In dataflow, a node (ECO) is ready for execution when it receives data on (some of) its input data channels. A general-purpose RTOS will typically schedule processes based on their priorities, ignoring data channel activities. Some RTOSes also support event-based notification for individual mailboxes or message queues. When using such kernels for dataflow, however, the user may be forced to check the status of several incoming channels inside the ECO code to ensure that all necessary data is available before proceeding. Unlike other RTOSes, DARK uses both priority and the status of all incoming data channels, together with a set of firing rules that specify what combination(s) of incoming data the ECO is waiting on. Thus, a thread starts executing only when all necessary data is ready.

DARK makes scheduling decisions using the firing rules associated with each ECO. Fig. 3-12 illustrates the structure of a firing rule, which is an array of one or more records consisting of a `firing_mask` and a new `priority` as the fields. The firing mask is a binary mask that specifies the input data channels that, if filled, should trigger the ECO to wake up (change to ready state). For example, the firing mask 00000111 indicates that the ECO is ready to fire when it has data on channels 0, 1, and 2.

```
typedef struct
{
        unsigned int  firing_mask;
        short int     priority;
} Priority_Firing_Mask;
typedef Priority_Firing_Mask* Firing_Rule;
```

Fig. 3-12 A firing rule.

An ECO can have more than one firing mask associated with it. For example, it can take one action when it has data in three incoming channels, while it can take another action when it has data in only two incoming channels. The application designer statically arranges the firing masks in the firing rule in order of their priorities. The `priority` field associated with a firing mask is the new priority that is assigned to the thread if the ECO is triggered as a result of the corresponding firing mask. To support efficient mask testing, the current status of all input channels is maintained in the form of another bit mask in the `in_ports_ready` field of the `TCB`. Every data channel read or write operation updates the corresponding bit of this mask. The implementation of the scheduler will be discussed in detail in Chapter 5.

3.4.2.2.2 Thread Management

The state of a thread is defined in part by the current activity of that thread. In DARK, a thread can be in one of the following six states at any time: `ready`, `blocked`, `wait_for_fire`, `timed_wait`, `timed_wait_for_fire` and `dead`. Fig. 3-13 illustrates this state model. When the kernel starts, each thread is in the `wait_for_fire` state because it is waiting for data. When a thread fires, it goes into the `ready` state. A thread blocks when it tries to read from an empty data channel or write to a full data channel. Once an ECO finishes processing incoming data, it calls the `wait_to_fire()` OS operation to enter the `wait_for_fire` state until more data comes along. An ECO can also delay execution for a pre-determined time, which puts the ECO into `timed_wait` state. The `timed_wait_for_fire` state is a combination of `wait_for_fire` and `timed_wait`. If an ECO is in this state, it will awaken either when the specified time expires or when its firing rule is triggered. Finally, an ECO enters the `dead` state when it finishes execution.

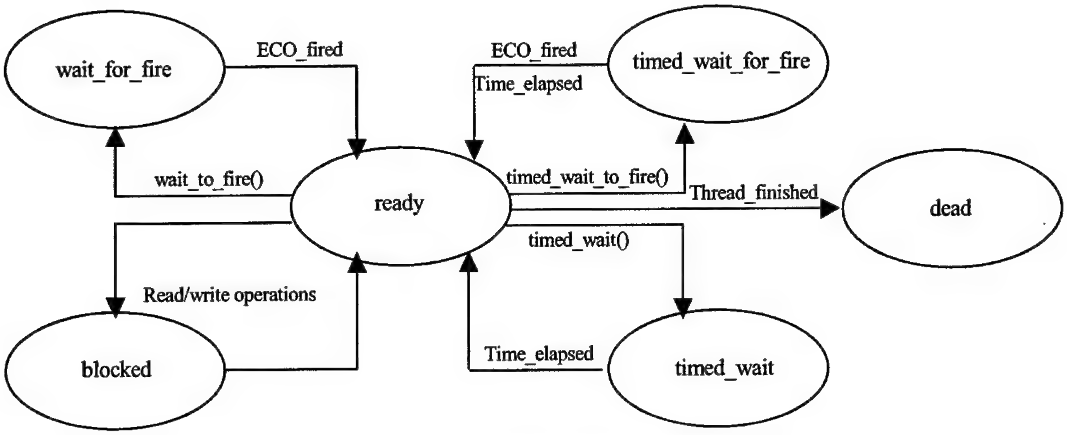


Fig. 3-13 Thread state diagram.

Due to the data-driven nature of dataflow processes, most of the thread management is done through operations carried out with every data channel read and write call. A read operation will unblock a thread waiting to write, as well as update `in_ports_ready`. A write operation will unblock a thread waiting to read, update that thread's `in_ports_ready`, test the listening thread's firing rule, and make the thread `ready` if it is triggered. Because of this relationship, DARK calls the scheduler after each read and write operation. If a thread is fired after these operations, it is placed in the ready queue according to its priority. The scheduler is also called as part of other OS functions, including `wait_for_fire()`, `timed_wait()`, and `timed_-wait_for_fire()`, so that the system can switch to an alternate ready thread if necessary.

3.4.2.2.3 High Speed Context Switching

The scheduler for an operating system can run as a separate thread, or can be called by other threads passively. DARK uses the former approach and has an active scheduler because an active scheduler aids in preemptive scheduling. Like most RTOSes, DARK saves and restores only selected registers during a context switch. The registers that are not used by the C run-time environment are not saved, so use of assembly language in ECO code requires special care. Most context switches in a dataflow application occur between the scheduler and executing threads. Minimizing the cost of such switches will increase performance. DARK supports the use of a dual-register-set architecture to support high-speed context switching between the scheduler and application threads, reducing the switching time by 80%.

Many digital signal processors used in embedded control have two register sets for increased performance. For example, the Analog Devices SHARC 21062 microprocessor on which DARK was originally implemented has a primary set and an alternate set of registers. DARK uses the primary register set for the kernel while the secondary set is used for executing application threads. Switching between the scheduler and the application involves simply flipping one bit in a control register and saving/restoring some key status registers. This approach leads to a drastic decrease in the context switching time between the DARK scheduler and application threads, as seen in Fig. 3-14.

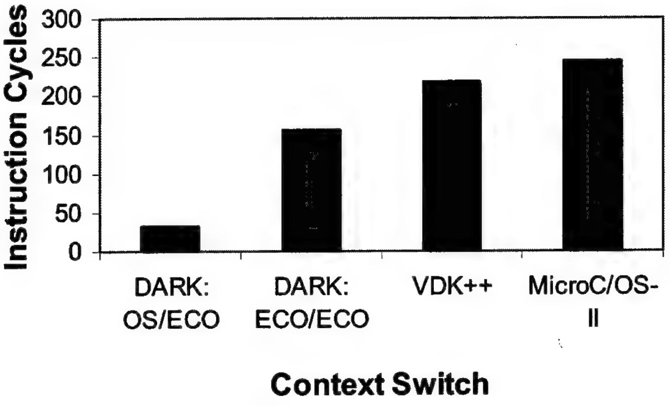


Fig. 3-14 Comparison of context switching times.

DARK's scheduler-to-application context switch is five times faster than the context switch that occurs between application threads. Figure 3.5 also shows the context switching times taken by two other commercial RTOSes (on the same processor).

3.4.2.2.4 Time Management

DARK allows an ECO to request a timed delay. In other RTOSes, the kernel typically checks each waiting thread at every clock tick, and adds it to the ready queue when the waiting period has expired. This technique can introduce unnecessary cost if there are more than a tiny number of waiting threads.

DARK uses a different approach to handle timed delays. When the `timed_wait()` or `timed_wait_for_fire()` function is called, the delay is converted into an absolute time by adding the current system time to it. Then, it is stored in the TCB. Next, the thread is then added to a (circular) waiting queue arranged in ascending order by absolute time. When a thread is added to the waiting queue, a kernel variable `actions_pending` is set to indicate that the scheduler should check the waiting queue. Each time the DARK scheduler is called, it compares the system time with the thread wakeup time of the first thread in the waiting queue. The thread is added to the ready queue if the waiting time has expired. The scheduler need only check the first thread in the waiting queue unless that thread's wait is over.

3.4.2.2.5 Interrupt Handling

Many portable RTOSes rely on a compiler-provided mechanism for interrupt handling where C functions can be used as interrupt routines. In this general approach, all registers are saved and restored while handling interrupts, increasing the overhead of interrupt-oriented context switching. The compiler provided by Analog Devices for its SHARC DSPs supports this approach, but also provides a second option of using the alternate register set for interrupt handling. Since the C runtime uses only the primary register set, this option works well for most RTOSes, allowing speedy interrupt handling. DARK cannot use this option, however, because it uses both the alternate and primary register sets on this platform.

DARK uses an alternative approach for handling external interrupts that provides performance comparable to this second option. Rather than placing actions directly in the interrupt handler itself, DARK uses a small-footprint handler that simply logs incoming events into an "event queue", which is managed by the DARK scheduler thread. The interrupt handler runs in the currently active register set and only needs to save and restore a couple of registers. It logs a 32-bit code representing the interrupt that was received into a circular buffer of incoming events, then returns control to the dispatcher. This queue of events is translated into messages sent on data channels to the application inside the dispatcher when it checks for `actions_-pending`. After that the dispatcher selects the highest priority thread to run.

In addition to the general-purpose interrupts explained above, DARK also supports clock interrupts and non-maskable interrupts (NMI). The clock interrupt ISR is written in assembly and simply increments the kernel variable `current_time` that is used for time management. Only a few registers required for incrementing a variable are saved and restored in this ISR. The NMI interrupt is used for emergency condition notification and requires a time critical response. In most cases, it results in a call to the application's emergency shutdown procedure, bypassing all other DARK and application code.

3.4.2.2.6 Mutual Exclusion

The most common approach to supporting inter-process synchronization in an RTOS is to provide semaphores. Other mechanisms that are often supported include event notification mechanisms and message queues. Though semaphores are powerful and are provided by most RTOSes, they can lead to a heavy cost in terms of performance. Another approach for mutual exclusion adopted by RTOSes is to disable interrupts when entering a critical section and reenable them on exiting the section. This option is unsuitable for large critical sections because of the increasing probability of missing important interrupts.

Fortunately, the unique aspects of dataflow processes ensure that most mutual exclusion problems do not arise in DARK. Threads share no resources or memory and communicate only through the data channels. The only conflicts that can arise are when two ECOs attempt to access the same data channel, or when an ECO and the scheduler thread both attempt to access an internal OS data structure. In effect, this means that all potential conflicts occur only when a thread is making an OS API call. The `in_OS_call` field of the thread's `TCB` is used as a simple flag to indicate when an application thread is in the process of making such a call. This field is only written by the ECO and only read by the scheduler thread. Since the scheduler is always called at the conclusion of each OS operation, if it is ready to perform a context switch on a running thread where `in_OS_call` is set, it can simply resume the thread—which will then finish the call, and promptly return control to the scheduler so it can be switched out. This forms a crude form of "safe points" for context switching. Otherwise, no mutual exclusion mechanisms are needed to manage dataflow applications. Note that interrupt handling can happen at any time, since there can be no conflicts between the interrupt handlers and application code.

3.4.2.2.7 Real-time Support

DARK provides options to users to selectively include real-time features in the kernel. The optimal fixed priority algorithm is shown to be the rate monotonic priority assignment (RMA) in which a task with a shorter period is given higher priority than a task with a longer period [vii]. The deadline driven scheduling algorithm is an optimal dynamic scheduling algorithm [vii]. But the dynamic real-time scheduling algorithms carry a lot of overhead, which may affect the system performance in a negative way. In case of deadline driven scheduling algorithm, the deadlines are monitored at each clock tick to assign the highest priority to the task with nearest deadline.

DARK provides support for using the fixed priority real-time scheduling algorithms. Along with the DFG definition, the user is given option to specify a statically scheduling algorithm through a function handle. If the function handle is assigned null, the default scheduling provided by DARK is used. DARK provides an implementation of the fixed priority RMA that can be optionally used by the application designer to assign priorities according to the rate monotonic approach. RMA uses the information in the DFG to calculate the priorities and exits without starting the application if a feasible schedule cannot be found.

In addition to this, DARK provides a simple API to applications for monitoring their real-time deadlines. In the interest of DARK's high-performance objective, the complex real-time support necessary for POSIX compliance has been avoided. Moreover, other POSIX-required features are omitted, including naming, file systems and signals. An entity can set a deadline for its execution using the following API function:

```
void set_deadline(int time);
```

The variable `time` in this function specifies the time by which the entity has to finish its execution. Whenever this function is called, the `time` is converted to an absolute time by adding the current system time to it. The ECO is then added to a deadline queue similar to the waiting queue. Later, the ECO calls `check_deadline()` to ascertain its adherence to the deadline. This function removes the associated entry from the deadline queue and returns true if the ECO met its deadline. In addition to this, the DARK scheduler also checks the first entry of the deadline queue on each invocation. If it finds an entity that missed its deadline, it calls a user-provided handler.

The kernel also provides an option in `OS_cfg.h` to monitor the switching cycle deadlines, which are important for dataflow applications. One switching cycle consists of sensing and updating all of the necessary components in the power stage being controlled. When this option is enabled, the `set_deadline` is called at the beginning of each switching cycle, whereas `check_deadline` is invoked at the end of each switching cycle. If a switching cycle exceeds its deadline, a user provided handler is called that may cause an emergency shutdown.

3.4.2.3 DARK-Configurable Options

Four distinct versions of the DARK kernel can be obtained by selectively removing certain kernel features. Removal of the features leads to an increase in performance together with a concomitant reduction in run-time flexibility. The application designer can select the most appropriate DARK version for a given application's requirements. Table 3-1 lists the features in different versions. In addition to this, the data-channels can also be configured according to the needs of the applications.

The "full-featured" version of DARK, with nothing disabled, is a multi-threaded preemptive kernel. Threads are dynamically scheduled based on their firing rules and priorities. After every read and write operation, the scheduler is invoked to check for higher priority threads. A context switch takes place if a higher priority thread is ready. The kernel also ensures fair scheduling between equal-priority threads. Please note that the fair scheduling here does not mean fair CPU-time slicing. It means that the DARK scheduler selects the next thread of equal priority and suspends the current thread whenever it is invoked.

The "non-preemptive" version of DARK does not invoke the scheduler on every OS call. Instead, each thread runs until it suspends itself—typically because it is waiting for more incoming data. Thus, the overhead of calling the scheduler after every read and write operation is avoided by sacrificing immediate response to higher-priority threads.

The remaining two versions of DARK are single threaded. They avoid the time spent in context switching, thereby giving a performance boost to the application. Such an approach may be unsuitable for an application with dynamic behavior, but may be ideal for monotonic applications that execute sequentially. The dynamically scheduled, single-threaded version of DARK continues to use firing rules and priorities to select which process is ready for execution. The statically scheduled, single-threaded version of DARK is fastest; it uses a precomputed firing order for threads, eliminating all use of priorities and firing rules.

DARK also provides a configurable option for data-channels. The data-channels in a dataflow application can be synchronous or asynchronous. Synchronous channels never contain more than one data item throughout the application execution. In other words, whenever a data item is written, the sink ECO consumes it before any subsequent write into the channel. Asynchronous data channels can contain multiple data items. Many dataflow applications use synchronous channels. The kernel provides a mailbox option, which can be used with any of the four DARK versions listed above with no change to the application code. The mailboxes avoid the costly queue maintenance operations and thus, provide a better performance.

In addition to these options, DARK also provides options for adding real-time features as discussed in the previous section. The user can select a fixed priority real-time scheduling algorithm such as RMA using a handler in the `DFG.h` file. The option of monitoring deadlines for each switching cycle can also be selected by the application designer.

Table 3-1 Configuration options available in DARK.

Versions	Features		
	Preemptive	Multithreade d	Dynamic scheduling
Full-featured DARK	X	X	X
Non-preemptive DARK		X	X
Dynamically-scheduled single-threaded DARK			X
Statically-scheduled single-threaded DARK			

3.4.3 Kernel Evaluation

In this subsection, we discuss the results obtained during the performance evaluation experiments of DARK. The experiments were conducted using an Analog Devices-SHARC 21160 80 MHz microprocessor. The Analog Devices VisualDSP++ 2.0 emulator was used to run the experiments and collect the data.

As described before, DARK can be used in any of the four modes using the configurable options. The experiments in this chapter exhibit the relative decrease in the kernel overhead through reducing the kernel features. The results show that the user can reduce the kernel overhead by more than 80% by the removal of features.

We have compared the performance of DARK with two kernels-MicroC/OS-II [viii] and Analog Devices VDK++ [ix]. MicroC/OS-II is a simple high-performance kernel written in C, which imposes certain limitations on the applications to increase the speed. VDK++ is a special kernel designed by Analog Devices to support their microprocessors. VDK++ is written in C++ and it was not found suitable for running high-performance applications after our experiments. The results of the performance comparison indicate that the DARK with all features enabled is more than 80% faster than Analog Devices' VDK++. The non-preemptive version of DARK is found to be more than 11% faster than the non-preemptive MicroC/OS-II. The main features of the kernels are compared with DARK in Table 3-2.

Table 3-2 Comparison of major properties of the kernels

Properties	Kernels		
	Full-featured DARK	**MicroC/OS-II**	**VDK++**
Implementation Language	C	C	C++
Preemptive	Yes	Yes	Yes
Multitasking	Yes	Yes	Yes
Upper limit on number of user tasks	None	56	None
Inter-task communication support	Yes	Yes	No
Call to scheduler	After API call	After API call	After API call
Mutual exclusion for interrupt handlers	NOT required	Required	Required
GUI for task initialization	None	None	Yes
Reconfigurability	Yes	None	Limited

3.4.3.1 Overview of Applications

The dataflow applications for power electronics controllers discussed in the previous chapter are used in conducting the performance evaluation experiments. All the three applications are developed as a part of PEBB project at Center for Power Electronics Systems, Virginia Tech. This Section contains an overview of the relative load that the applications impose on the kernel.

The Table 3-3 summarizes the key categories of Kernel API calls made by the three test applications. The numbers shown represent how many times each type of operation is performed in one switching cycle of the application. One switching cycle consists of sensing and updating all of the necessary components in the power stage being controlled. For the test applications discussed here, that amounts to executing each ECO in the application exactly once.

Table 3-3 Kernel operations per switching cycle.

Operations	Open loop inverter	Closed loop inverter	Boost rectifier
Read from data channel	9	20	29
Write to data channel	9	20	29
App-App context switches	6	8	17
App-kernel context switches	14	18	36
Ready queue insertions	7	9	18
Ready queue deletions	7	9	18

The dataflow graphs for these applications are given in 3.3. The dataflow application for open loop inverter is the simplest containing only seven ECOs and nine data channels. The closed loop inverter consists of nine ECOs and twenty data channels. This application contains one current loop, which is completed by the hardware sensors that take the output results from the PEBB drivers and control the application via interrupt driven data channels. The boost rectifier application contains two loops—current loop and voltage loop, which again are controlled by the external hardware sensors. These loops are called feedback loops as they give a feedback to the applications based on their pervious outputs. The dataflow application for the boost rectifier contains eighteen ECOs and twenty-nine data channels.

3.4.3.2 DARK Versions

DARK provides options for selectively removing the features for gaining performance. In this section, we will compare the relative performance gains as features are removed. Experiments were conducted on all of the eight DARK options possible—Preemptive (Full-featured) DARK, Non-preemptive DARK, Dynamically scheduled single threaded DARK and Statically scheduled single threaded DARK, with or without the mailbox option. The mailbox option, which is suitable for synchronous communication, provides the best performance.

Table 3-4 shows the performance comparison figures obtained when message queues are used whereas the figures for the mailboxes are provided in Table 3-5. As shown, the full-featured version of DARK takes 4203 instruction cycles to complete one switching cycle of the application, whereas the minimal featured version with mailboxes takes only 514 instruction cycles. A significant performance gain is achieved by removing the multi-threaded feature, as the overhead for context switching diminishes drastically. In addition, static scheduling also leads to performance gain, because the ready queue and firing rule related operations are completely eliminated. The mailbox option eradicates the queue maintenance operations, because of which it is faster than the versions with message queues. The Fig. 3-15 shows a graphical representation of the reduction in kernel overhead, both in message queues and mailboxes versions.

Table 3-6 and Table 3-7, and Fig. 3-16 show the similar comparison results for the closed loop application. The full-featured DARK takes 6603 instruction cycles for completing one switching cycle of the application while the minimal featured version takes 1073 instruction cycles.

At the end, we have provided the results obtained when the boost rectifier application was used to run the experiments. The data in the Table 3-8 and Table 3-9 show that the full-featured DARK takes 12371 instruction cycles for executing one switching cycle of the application whereas the minimal-featured version takes only 1561 instruction cycles. The reduction in kernel overhead by removing features is shown in Fig. 3-17.

Table 3-4 Performance of DARK versions for the open loop application with message queues

Operations	Execution time for one switching cycle (Instruction cycles)			
	Full-featured	Non-preemptive	Single-threaded dynamically scheduled	Single-threaded statically scheduled
ECO execution	235	235	235	235
Scheduling	529	529	219	212
Context switching	1148	1148	0	0
Ready queue operations	525	525	525	0
Data channel operations	1304	999	945	312
Other OS operations	462	460	77	0
TOTAL	**4203**	**3896**	**2001**	**759**

Table 3-5 Performance of DARK versions for the open loop application with mailboxes

Operations	Execution time for one switching cycle (Instruction cycles)			
	Full-featured	Non-preemptive	Single-threaded dynamically scheduled	Single-threaded statically scheduled
ECO execution	235	235	235	235
Scheduling	529	529	219	212
Context switching	1148	1148	0	0
Ready queue operations	525	525	525	0
Data channel operations	1040	734	734	67
Other OS operations	462	450	121	0
TOTAL	**3939**	**3621**	**1834**	**514**

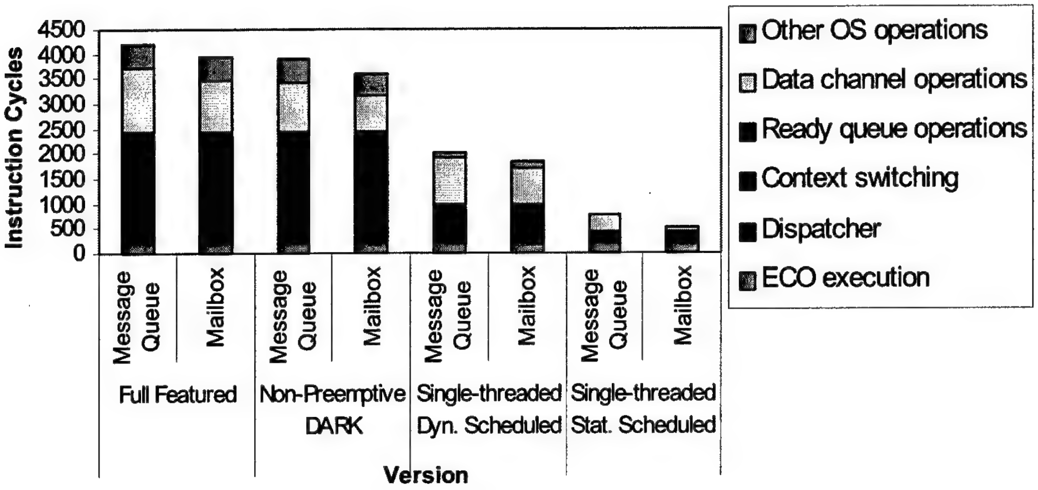


Fig. 3-15 Performance of DARK versions for the Open-loop Inverter application.

Table 3-6 Performance of DARK versions for the closed loop application with message queues

Operations	Execution time for one switching cycle (Instruction cycles)			
	Full-featured	Non-preemptive	Single-threaded dynamically scheduled	Single-threaded statically scheduled
ECO execution	583	583	583	583
Scheduling	679	679	297	270
Context switching	1476	1476	0	0
Ready queue operations	612	612	612	0
Data channel operations	2659	2059	2059	731
Other OS operations	594	594	203	0
TOTAL	**6603**	**6003**	**3754**	**1584**

Table 3-7 Performance of DARK versions for the closed loop application with mailboxes

Operations	Execution time for one switching cycle (Instruction cycles)			
	Full-featured	Non-preemptive	Single-threaded dynamically scheduled	Single-threaded statically scheduled
ECO execution	583	583	583	583
Scheduling	679	679	297	270
Context switching	1476	1476	0	0
Ready queue operations	612	612	612	0
Data channel operations	2271	1698	1698	220
Other OS operations	594	594	181	0
TOTAL	**6215**	**5642**	**3371**	**1073**

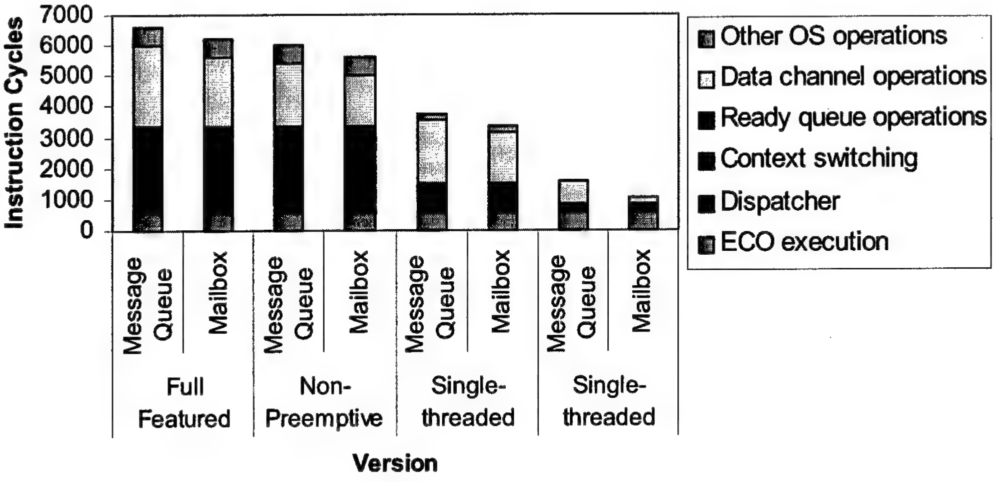


Fig. 3-16 Performance of DARK versions for the Closed-loop Inverter application

Table 3-8 Performance of DARK versions for the boost rectifier application with message queues

Operations	Execution time for one switching cycle (Instruction cycles)			
	Full-featured	Non-preemptive	Single-threaded dynamically scheduled	Single-threaded statically scheduled
ECO execution	778	778	778	778
Scheduling	1354	1354	585	567
Context switching	2952	2952	0	0
Ready queue operations	1811	1811	1811	0
Data channel operations	4252	3174	3174	1020
Other OS operations	1224	1224	205	0

Table 3-9 Performance of DARK versions for the boost rectifier application with mailboxes

Operations	Execution time for one switching cycle (Instruction cycles)			
	Full-featured	Non-preemptive	Single-threaded dynamically scheduled	Single-threaded statically scheduled
ECO execution	778	778	778	778
Scheduling	1354	1354	585	567
Context switching	2952	2952	0	0
Ready queue operations	1811	1811	1811	0
Data channel operations	3699	2640	2640	216
Other OS operations	1224	1224	88	0
TOTAL	**11818**	**10759**	**5902**	**1561**

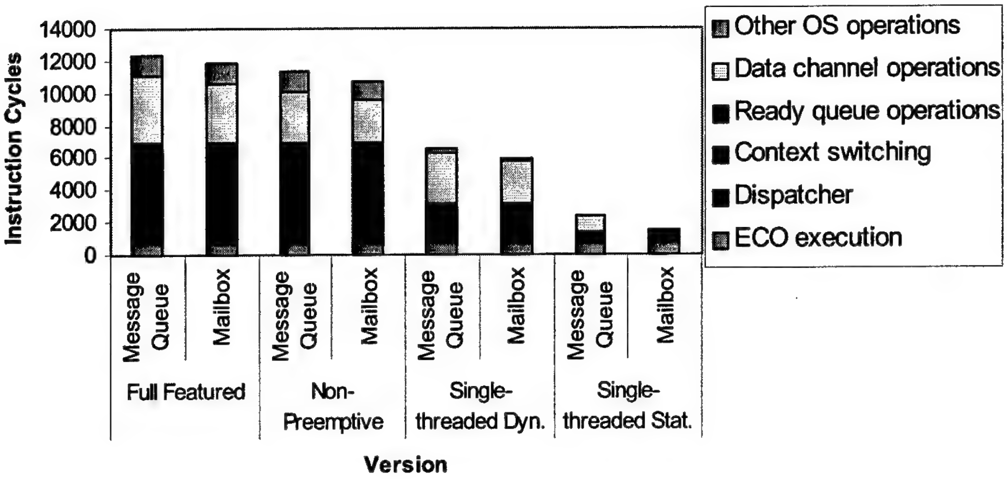


Fig. 3-17 Performance of DARK versions for the Boost Rectifier application

3.4.3.3 DARK vs. MicroC/OS-II

In this section we present the results obtained by the performance comparison of Full-featured DARK with Non-preemptive MicroC/OS-II [viii]. MicroC/OS-II is a preemptive multi-tasking kernel written by Jean J. Labrosse in C. It schedules tasks based on their priorities and does not provide any special support for dataflow applications. MicroC/OS-II provides API for inter-task communication using message queues. For experimentation purposes, the support for firing rules was added for dynamic scheduling of dataflow applications. Semaphores were used to avoid race conditions related to message queues, which are shared data structures.

MicroC/OS-II makes many simplifying decisions in order to provide high-performance. The number of user tasks or threads is limited to 56 and every task should have different priority. This enables the kernel to completely eliminate the costly ready queue operations. MicroC/OS-II uses bit tables for faster scheduling of tasks. On the negative side, it makes the fair scheduling of tasks impossible, as same priorities cannot be assigned to multiple tasks. The limit on number of tasks is also a concerning factor for dataflow applications because they often have more tasks than regular applications for increasing the reusability and modularity. In DARK, the number of threads is limited by the system memory and multiple threads can have the same priority, which enables fair scheduling.

Table 3-10 Performance of DARK versions for the boost rectifier application with message queues

Operations	Execution time for one switching cycle (Instruction cycles)			
	Full-featured	Non-preemptive	Single-threaded dynamically scheduled	Single-threaded statically scheduled
ECO execution	778	778	778	778
Scheduling	1354	1354	585	567
Context switching	2952	2952	0	0
Ready queue operations	1811	1811	1811	0
Data channel operations	4252	3174	3174	1020
Other OS operations	1224	1224	205	0
TOTAL	12371	11293	6553	2365

Table 3-11 Performance of DARK versions for the boost rectifier application with mailboxes

Operations	Execution time for one switching cycle (Instruction cycles)			
	Full-featured	Non-preemptive	Single-threaded dynamically scheduled	Single-threaded statically scheduled
ECO execution	778	778	778	778
Scheduling	1354	1354	585	567
Context switching	2952	2952	0	0
Ready queue operations	1811	1811	1811	0
Data channel operations	3699	2640	2640	216
Other OS operations	1224	1224	88	0
TOTAL	11818	10759	5902	1561

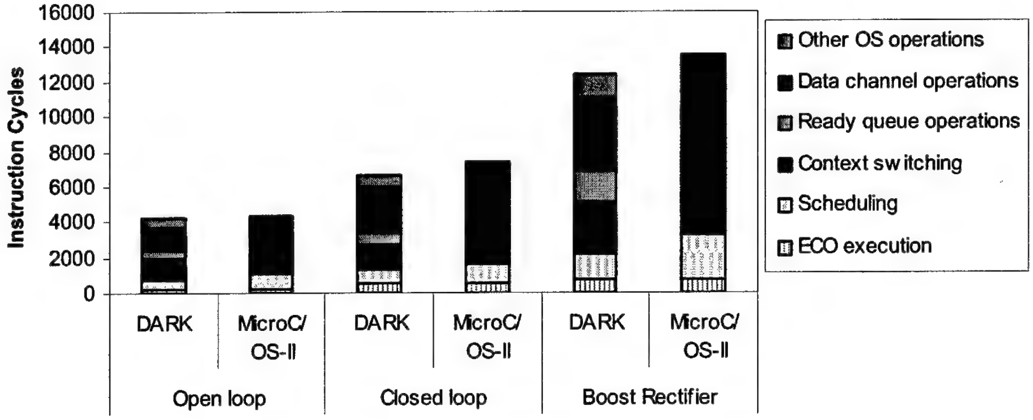


Fig. 3-18 Performance comparison between Full-featured DARK with message queues and MicroC/OS-II.

Unlike DARK, MicroC/OS-II does not have typed data channels. The data channels of MicroC/OS-II support only void pointers, thus reducing the type safety. This also increases the possibility of memory leaks and dangling pointers. Moreover, MicroC/OS-II does not provide any special scheduling for dataflow applications based on the data in their incoming data channels. This increases the burden on the application designers who in turn are forced to use an invariant of static scheduling for their applications.

At last, the preemptive scheduling approach adopted in MicroC/OS-II is an unnecessary overhead for dataflow applications. In a dataflow application, a thread becomes *ready* only after some specific API functions such as read or write to data-channel or after interrupts. So, it is not necessary to check for higher-priority threads at each clock tick as is done in MicroC/OS-II. Instead this process should occur only after possible operations that can make a dataflow thread ready, which is the approach adopted in DARK. So, the preemptive scheduling approach adopted in DARK is more effective and has lesser overhead than MicroC/OS-II, as it is designed specially for dataflow applications.

The Table 3-12 and Fig. 3-18 show the results obtained in the performance comparison of Non-preemptive DARK with message queues and Non-preemptive MicroC/OS-II.

Table 3-12 Performance comparison between Full-featured DARK with message queues and MicroC/OS-II

Operations	Open loop		Closed loop		Boost Rectifier	
	Non-Preemptive DARK	MicroC/OS-II	Non-Preemptive DARK	MicroC/OS-II	Non-Preemptive DARK	MicroC/OS-II
ECO execution	235	235	583	583	778	778
Scheduling	529	882	679	1013	1354	2467
Context switching	1148	1575	1476	2098	2952	4253
Ready queue operations	525	0	612	0	1811	0
Data channel operations	999	1684	2059	3674	3174	5958
Other OS operations	460	0	594	0	1224	0
TOTAL	**3896**	**4376**	**6003**	**7368**	**11293**	**13456**

3.4.3.4 DARK vs. Analog Devices-VDK++

VDK++ [ix] is a preemptive multi-tasking kernel written in C++. It is provided by Analog Devices as a component of its VisualDSP++ IDE for developing applications for its DSPs. VDK++ provides users the options of selecting the scheduling policy. It supports cooperative scheduling and round robin scheduling in addition to preemptive scheduling. The associated overhead because of object-oriented features makes VDK++ unsuitable for high-performance applications.

The scheduling adopted in VDK++ is similar to DARK. The scheduler is called after an API call to the kernel, which can make a higher priority thread ready. VDK++ does not provide any support for inter-task communication, which is an integral part of any kernel. For performing the experiments, we added data channel support to VDK++ similar to DARK. While accessing the data-channels, we had to use semaphores in VDK++ to avoid shared data problems. For dynamic scheduling of the dataflow applications, the event signal API provided by the VDK++ kernel was used in read and write operations.

VDK++ provides a user interface for the programmers through which the properties of the kernel can be changed. The user interface is also used to add or remove events, semaphores, threads, etc. This makes the kernel easy to use but on the hand, also increases the inflexibility of the kernel. It is a drawback for the dataflow applications, which rely heavily on reusability of individual ECOs because the firing rules of existing ECOs cannot be used in a new application. The user needs to create a separate set of events for each project or application. Also, as the kernel files are generated automatically by the kernel for every project, modifying VDK++ to make it suitable for dataflow applications is very difficult and expensive. Moreover, the kernel operations in VDK++ are very heavyweight and are unsuitable for applications that require high-performance. We conclude that VDK++ is suitable for those applications that have independent threads and where the speed of execution or the kernel overhead is not a big constraint. Table 3-13 and Fig. 3-19 show the results obtained in the comparison of Full-featured DARK with VDK++. As seen in the figures, the use of expensive VDK++- semaphore and event operations in data channel reads and writes increases the cost by a tremendous amount of instruction cycles.

Table 3-13 Performance comparison between Full-featured DARK with message queues and Analog Devices- VDK++

Operations	Open loop		Closed loop		Boost Rectifier	
	Full-featured DARK	VDK++	Full-featured DARK	VDK++	Full-featured DARK	VDK++
ECO execution	235	235	583	583	778	778
Scheduling	529	2243	679	3046	1354	5564
Context switching	1148	1659	1476	2312	2952	4278
Ready queue operations	525	0	612	0	1811	0
Data channel operations	1304	14850	2659	33000	4252	47850
Other OS operations	462	2551	594	4093	1224	7461
TOTAL	**4203**	**21538**	**6603**	**43034**	**12371**	**65931**

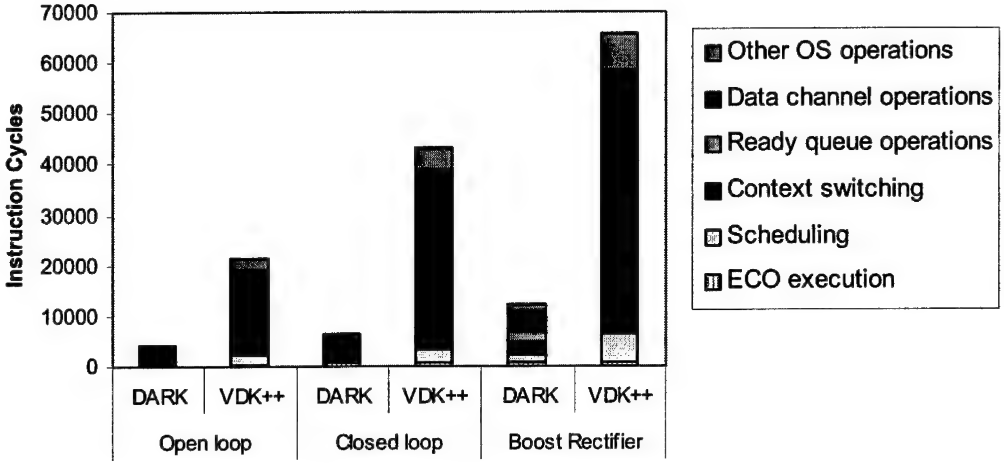


Fig. 3-19 Performance comparison between Full-featured DARK with message queues and Analog Devices-VDK++

3.4.3.5 Summary of Results

In this section, we have summarized the results obtained in all the experiments. The Fig. 3-20, Fig. 3-21, and Fig. 3-22 present the summaries for Open-loop, Closed-loop and Boost Rectifier applications, respectively. As shown in the figures, the configurable options of DARK present opportunities to application designers for selecting a tailored version that meets the performance requirements. The figures use the same legend as used in previous figures.

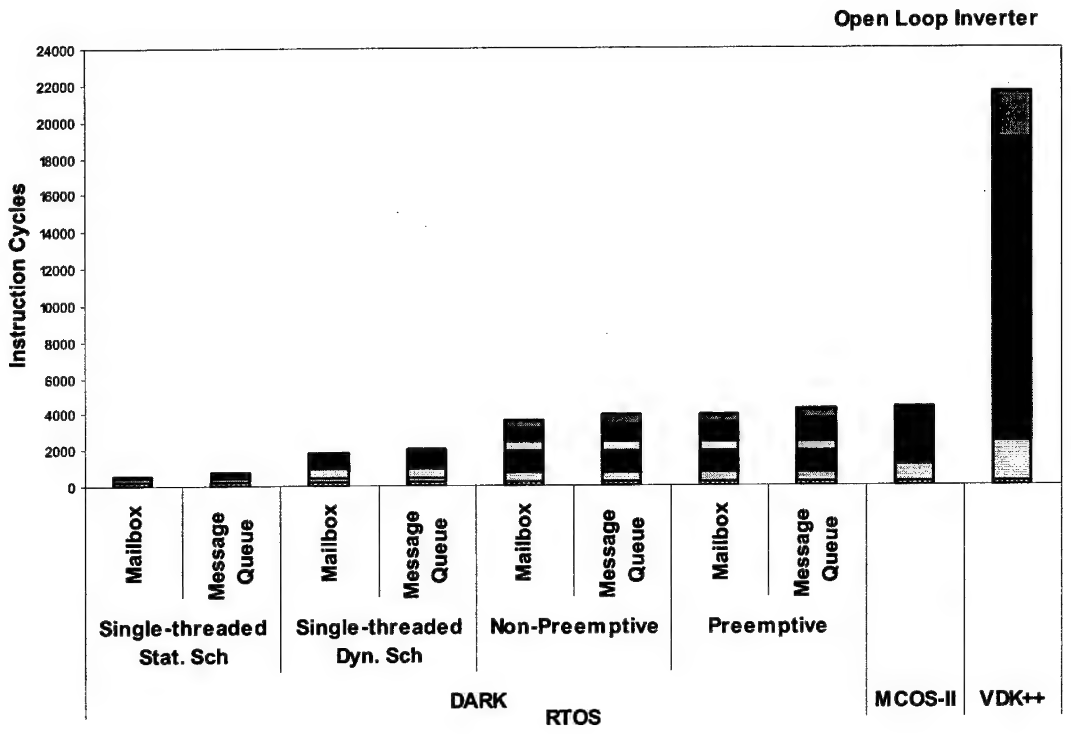


Fig. 3-20 Summary for results obtained while using Open-loop Inverter application.

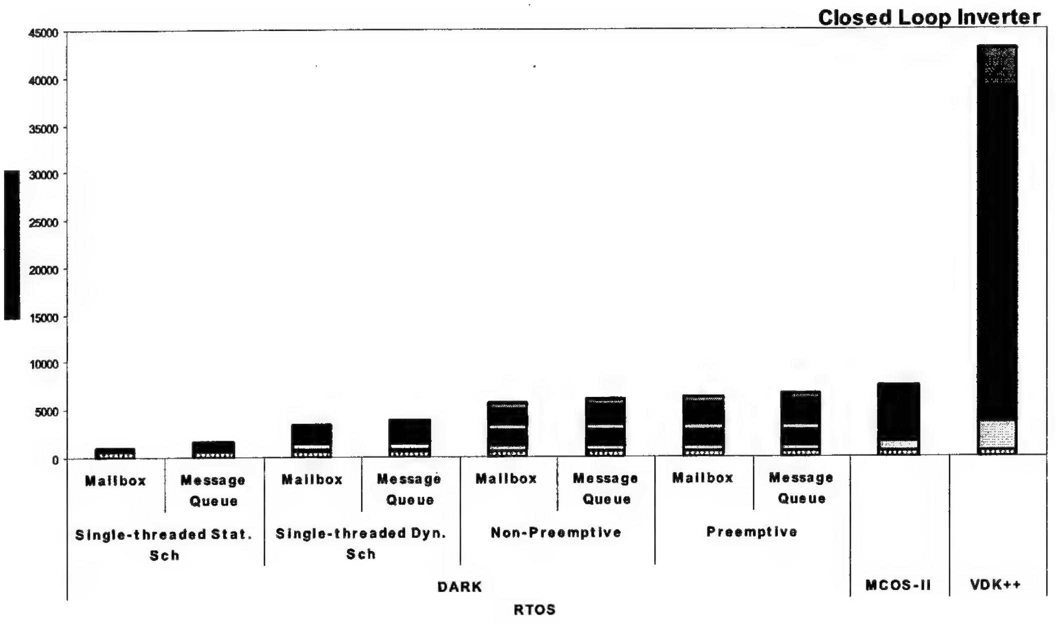


Fig. 3-21 Summary for results obtained while using Closed-loop Inverter application

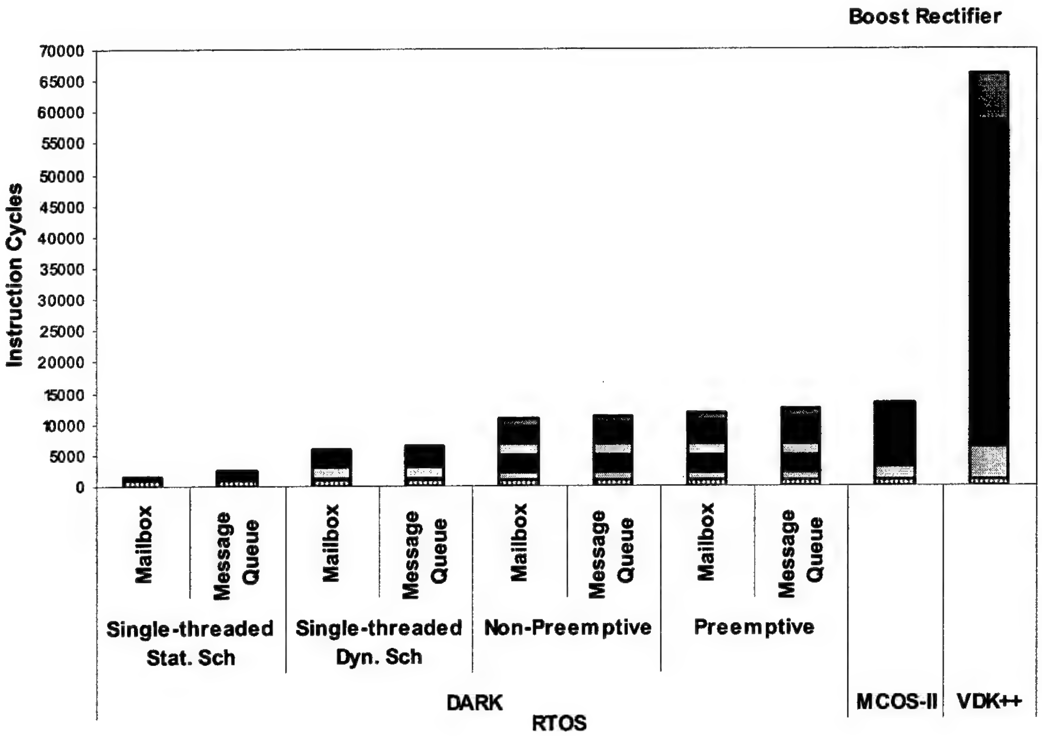


Fig. 3-22 Summary for results obtained while using Boost Rectifier application

3.5 DARK++

It is worth discussing the advantages that object-oriented programming offers that warrant its use. We discuss below, some important points in this regard:

- Compiler-enforced encapsulation in C++ offers more protection due to the notion of classes and different access levels for the various data members and behaviors - this applies to all OO systems in general, and therefore to our system as well.

- C++ offers more type safety with more static type checking – this is true of components that can be implemented by inheritance from a generic base class with parameterized subclasses for different data types, as opposed to having a generic module that is used for all data types with appropriate typecasting as is done in non-OO implementations generally. Typecasting makes the system more error-prone.

- OO systems are more naturally extendible by inheritance and overriding of base class methods in subclasses.

In our system, we found that with respect to the classically cited advantages of OO programming, viz., modularity, reusability and maintainability, our system is comparable to the non-OO system and does not offer any special gains because the non-OO system also has a well-defined structure with a standardized and reusable library of components.

3.5.1 DARK++ design overview

The DARK++ kernel has been designed and implemented as a platform to run PEBB systems, which are based on dataflow. The overall goal of PEBB-oriented research is to achieve modularization, standardization and reusability of power electronics components. The goal of DARK++ is to support this same modularity, standardization and reusability within the control software. Since the OO paradigm offers the advantages of software modularity and reusability, it is a natural fit with PEBB research. The DARK++ system comprises a library of ECOs that can be modified or extended very easily by using the classes and templates provided.

Due to the advantages of using OO programming for supporting PEBB systems, it is worth exploring its feasibility. Since we are dealing with real-time embedded systems, efficiency is a very important concern. We need to consider both the efficiency *requirements* in the given scenario, and also the efficiency *issues* that we need to overcome due to the use of OO programming.

Dataflow applications are comprised of processes that are essentially data driven. This necessitates causal ordering of the processes, which is done by the scheduler. However, for a real-time system, processes also have to be temporally ordered since every process has to meet its real-time deadline. In such a scenario, the overhead that the OS causes plays a crucial role. Moreover, dataflow applications typically comprise a larger number of smaller processes as a result of striving for modular, independent and reusable components. The natural consequence of a large number of processes is:

- high overhead in scheduling and context switching
- high frequency of inter-component communication

Over and above these efficiency issues, the OO paradigm introduces additional performance issues. The main sources for concern over extra run-time overhead are:

- Heap-allocated memory, due to the extensive dynamic creation of objects
- Dynamic binding, due to the use of virtual methods

- Method call overhead, due to a large number of small methods in classes

3.5.2 Kernel Architecture

This chapter describes the design of DARK++. It provides insight into the features an application designer needs to understand in order to write ECOs and use DARK++ for an application. It also discusses the rationale behind the most critical design decisions. This chapter presents the most important classes in the DARK++ system, including their responsibilities, interactions and the interfaces they provide. The following section is a discussion of the features of dataflow applications and the requirements that the dataflow model imposes on the kernel.

3.5.2.1 DARK++

DARK++ is a high-performance object-oriented kernel that addresses the requirements imposed by dataflow. It aims to reduce the overhead due to the use of OO programming. DARK++ is a preemptive, multi-threaded kernel. It always runs the highest priority thread that is ready. An executing process is preempted if a higher priority process is found to be ready when the currently running process makes a call to a kernel API. DARK++ implements efficient context switching by taking advantage of *dual-register-set* hardware and saving and restoring only the required registers as opposed to all of them whenever this is feasible. DARK++ also provides support for dynamic priorities, firing rules for specifying the data channel conditions necessary for process wakeup, and typed data channels for efficient and reliable inter-process communication. DARK++ is implemented in C++, with a few key elements- context switching, dual register set support, and interrupt handling written in assembly. Because it is intended for embedded power electronics control, it currently runs on Analog Devices SHARC 21xxx 32-bit digital signal processors. Dataflow processes, or ECOs, are implemented as C++ classes. DARK++ uses a statically initialized set of ECO objects, together with a statically initialized set of interconnecting data channel objects.

3.5.2.1.1 DARK++ Classes

The major classes that comprise the DARK++ system are: `DARKpp`, `ECO`, `Data_Channel` and some helper classes that form kernel's internal data structures. The `DARKpp` class is a singleton, which means there is only one DARKpp object (the kernel object) in the system. It is responsible for scheduling and running the processes, which are represented as `ECO` objects. The `ECO` class is a base class from which templates for specific ECOs are derived.

These templates take the types of their input ports as parameters. Since there are two types of data channels – the queued and the mailbox, which are in turn implemented as templates derived from the base Data_Channel class, the input port type parameters for specific ECOs help specify which of the two types of data channel each port of an ECO is. This can be tailored to suit the specific application; i.e., the application designer can instantiate an ECO appropriately. The ECOs read data from their input data channels, perform necessary computation and write data to their output data channels. The data channels, on receiving data, fire the next process, by adding their sink ECO to the ready queue, which is an internal data structure that the system uses.

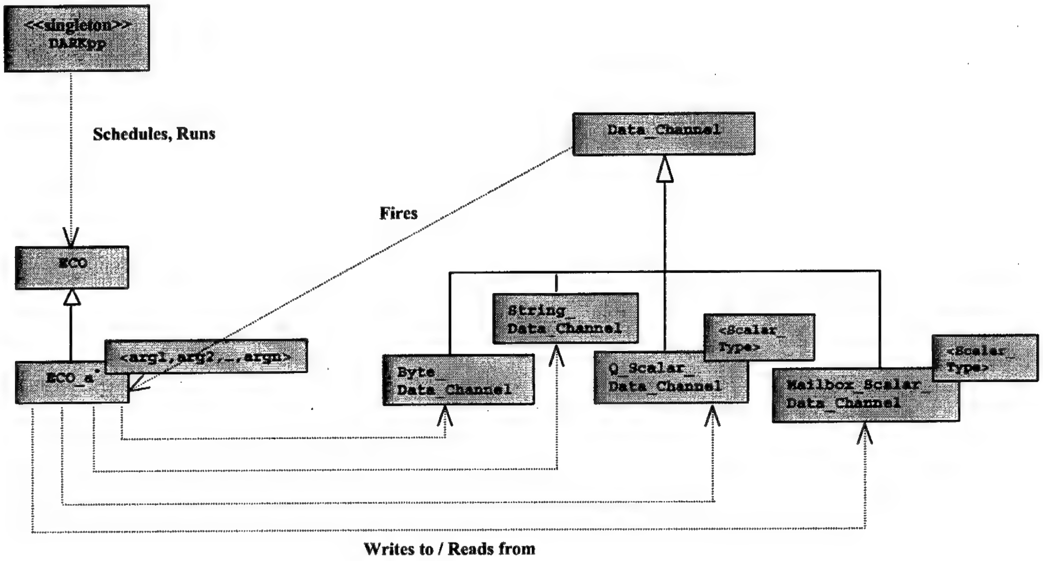


Fig. 3-23 DARK++ System Diagram.

Fig. 3-23 gives the "big picture" of the system. It shows the most important relationships among the main classes and the interaction amongst the respective objects.

3.5.2.1.2 Class DARKpp

The DARKpp class is the kernel class. It performs scheduling of the processes and makes use of the *ready queue* and *event queue* data structures, which are also implemented as classes (Ready_Queue, Event_Queue respectively).

The kernel, in its every iteration, checks for pending interrupts and handles them if there are any. It then executes the next ready process from the ready queue if this process has a priority greater than or equal to the currently running process. Thus, it preempts the current process if it finds a new process of greater or equal priority. Preemption on encountering an equal priority ensures fairness by preventing a process from hogging the processor for a long period.

The kernel uses the variable `actions_pending` whenever it needs to check whether any timed events have to be performed/pending interrupts have to be serviced. This variable can have one of three values:

- `no_actions`, which indicates that there are no pending timed events/interrupts
- `future_actions`, which indicates pending timed events
- `current_actions`, which indicates pending interrupts

Since there has to be a single kernel object in the system, a static method that returns a static reference to a kernel object is used. This method, `getInstance()`, is shown in Fig. 3-24.

```
DARKpp& DARKpp :: getInstance()
{
        static DARKpp the_kernel;
        return the_kernel;
}
```

Fig. 3-24 The getInstance() method.

3.5.2.1.3 Class ECO

The `ECO` class is an abstract base class for ECOs containing the ECO implementation method as a virtual method, which can be defined in the particular ECO subclass, since ECOs are functionally distinct. The ECO objects in the system are the processes that are scheduled and executed by the `DARKpp` kernel.

An ECO designer should take the following steps to write a class for a specific ECO:

- Call the base `ECO` class constructor from the constructor of the new class with the following parameters: number of input and output ports, the firing rule for the ECO, the initial priority, an array of pointers to the input ports and an array of pointers to the output ports.
- Set the specific configuration information for the ECO in the new class's constructor.
- Write the implementation function and any other new functions if required.

Figure 3.3 shows the ECO class interface. It includes comments that explain the role of every data member/method.

```
class ECO
public:
Implementation() = 0; // pure virtual function
Wait_To_Fire();      //returns true or returns to OS
Register_OS();                 // registers with DARKpp
Current_Priority();  // returns current ECO priority
SetPriority(Priority ); // sets ECO priority
SetIn_Ports_Ready(Firing_Mask );//set mask after write
Timed_Wait_To_Fire(int );
Delay(int );
In_Ports_Ready();        // returns mask showing ready ports
SetProcess_State(ProcessState );
Blocked();                // returns true if process is blocked
SetWakeup_Call(Firing_Mask );  // sets the mask that fires the ECO
Wakeup_Call();                       // returns the mask that woke it up
Process_State();                     // returns process state
FIRE_Rule();                         // returns firing rule
Get_Heap_Position();  // gets process's position in ready queue heap
Set_Heap_Position(int );// used to alter position of ECO in ready queue
swap(ProcessState );     // changes process state and goes to OS
ECO_Env();                // returns context information for the process
Stack();        // returns pointer to the stack in which process is running
StackSize();              // returns size of process stack

protected:
Register_As_Source(int ); // register as source ECO of o/p port(s)
Register_As_Sink(int );   // register as sink ECO of i/p ports(s)
```

Fig. 3-25 ECO Class Interface.

3.5.2.1.4 Class Data_Channel

The Data_Channel class is a base class for all types of data channels. From this class, we have a derived Q_Scalar_Data_Channel and Mailbox_Scalar_Data_Channel template classes. While Q_Scalar_Data_Channel represents queued data channels, which can contain multiple data items (specified in the constructor), while Mailbox_Scalar_Data_Channel represents mailbox data channels, which can contain a single data item. It is more efficient to implement mailbox data channels separately so that we can do away with queue arithmetic.

Besides these, the String_Data_Channel and Byte_Data_Channel classes are provided. If required, specific data channels for user-defined data types may be defined as subclasses of the Data_Channel class by using *traits*. The need for a Data_Channel base class and template subclasses for scalar data channels, rather than a single template base class arises because ECO objects need to store pointers to their input and output data channels (so as to *read/write* from the appropriate data channel). It would not be possible for an ECO object to contain an array of Data_Channel pointers if Data_Channel were defined as a template base class. This necessitates having a Data_Channel base class that represents any type of data channel.

As mentioned, the ECO objects store an array of *pointers* to `Data_Channel` objects rather than an array of `Data_Channel` objects. The rationale for this is that a data channel, essentially being an interconnection between two ECOs, forms the output data channel for one ECO and the input to another. Since both the *source* and the *sink* ECOs require a knowledge of the identity of this data channel, we would need this `Data_Channel` object in the *output data channel* array of the source ECO as well as in the *input data channel* array of the sink ECO, which is not possible. Therefore references to the `Data_Channel` objects (`Data_Channel` object pointers) are stored instead.

Data channels may be interrupt-driven or non-interrupt-driven. While interrupt-driven data channels are input data channels to processes that are fired by interrupts, the non-interrupt-driven data channels are written to by their source processes. This information (interrupt-driven or not) is specified in the constructor to the data channels.

Fig. 3-26 through Fig. 3-29 show the interfaces provided by all the data channel classes.

```
class Data_Channel
public:
Capacity();     // max. no. of entries
Entries();      // current no. of entries
Available_Capacity();
Flush(int ); // remove given no. of entries from end of queue
Blocked();      // returns true if blocked
protected:
Register_OS(); //registers with DARKpp
setBlocked();
resetBlocked();
/* Read_bytes pass char pointer to read, no. of bytes to be read */
Read_bytes(char* , int );
/* Write bytes pass char array to be written, no. of bytes */
Write_bytes(char* , int );
```

Fig. 3-26 Data_Channel Base Class Interface.

```
template Q_Scalar_Data_Channel<Scalar_T> :: public Data_Channel,
template Mailbox_Scalar_Data_Channel<Scalar_T> :: public Data_Channel
public:
Read(Scalar_T& );      // pass reference parameter of appropriate
// type to read
Write(Scalar_T );      // pass parameter of appropriate type to
// write
```

Fig. 3-27 Scalar_Data_Channel Template Class Interface

```
class String_Data_Channel :: public Data_Channel

public:
Read(char* );  //pass char pointer to read in string
Write(char* ); //pass char array write
```

Fig. 3-28 String_Data_Channel Class Interface.

```
class Byte_Data_Channel :: public Data_Channel

public:
/* Read_bytes follows: pass char pointer to read, no. of bytes to be
read */
Read_bytes(char* , int );

/* Write_bytes follows: pass char array to be written, no. of bytes */
Write_bytes(char* , int );
```

Fig. 3-29 Byte_Data_Channel Class Interface.

3.5.2.2 Client Code

Having discussed the important classes and the framework of the system, we now explain how these can be used by an application designer to run a dataflow control application.

We consider the closed-loop three-phase inverter as an example control application and show how it can be run using our kernel. Following is the dataflow graph for the application.

This application comprises 9 ECOs and 20 data channels. To run this application, the data channel objects first have to be declared. Following this, the ECO objects are declared with the association amongst them being established by providing the input and output port information in the constructors of the ECOs. The constructor takes the other necessary information as well, such as the configuration information for the ECO, the firing rule, the initial priority of the ECO and the size (in bytes) of stack space required to run the ECO.

Fig. 3-6 shows the template code for a sample ECO – the `Adc_Va`. As we can see, it takes the types of the input and output ports as parameters. The implementation body is a *while* structure that fires the ECO again if it has data in its input port. Based on which firing mask triggered the ECO (`wakeup_call`), the appropriate action is performed. These actions are also provided for the ECO.

```
template <class Bool_Data_Channel_i, class Float_Data_Channel_o >

void Adc_Va <class Bool_Data_Channel_i, class Float_Data_Channel_o>

:: Implementation(){            do
          {              switch (wakeup_call)
                         {
                            case ADC_VA_FIRING_MASK_DEFAULT: default_action(); break;

                              case ADC_VA_FIRING_MASK_EXCEPTION: exception_handling(); break;
                         }
          }while (Wait_To_Fire());
```

Fig. 3-30 Template for ECO Adc_Va.

```
void default_action()
{              /* Input variable */
bool start;

/* Output variable */
float va;

int Adc_offset;
float Adc_scale;
float Va_offset;
float Va_scale;

/* Intermediate variable */
int Adc_value;

Adc_offset = confg.Adc_offset;
Adc_scale = confg.Adc_scale;
Va_offset = confg.Va_offset;
Va_scale = confg.Va_scale;

/* Read input from data channel */
Adc_Va_Start->Read(start);

Adc_value = *(int *)confg.Data_buffer;
Adc_value -= Adc_offset;

va = (float)(Adc_value) * Adc_scale;

va = (va-Va_offset) * Va_scale;

va = 60.0;

/* Update output data channel */
Adc_Va_Va->Write(va);

}
```

Fig. 3-31 Default action for ECO Adc_Va

3.5.2.3 DARK++ Kernel Features

3.5.2.3.1 Thread Management

An ECO can be viewed as a process that executes its `Implementation` code provided by the ECO designer. The various possible states of these processes are: *ready*, *run*, *blocked*, *wait_for_fire*, *timed_wait*, *timed_wait_for_fire* and *dead*. When the kernel starts, each thread is in the *wait_for_fire* state. A process is in *ready* state once its required input data channels have data tokens in them. The ready process of the highest priority is run by the kernel and such a process (an executing process) is in the run state. The process is *blocked* when it tries to write to a full data channel.

After every *read* operation on a data channel, the status of the source ECO (ECO that writes to this data channel) is checked. If the source ECO is found to be blocked, then it is unblocked. Similarly, after every *write* operation, the mask of its sink ECO (ECO that reads from this data channel) is updated; i.e., the bit corresponding to the data channel in question is set. Thus, while a *read* operation could unblock a process blocked on a data-channel, a *write* operation could fire it.

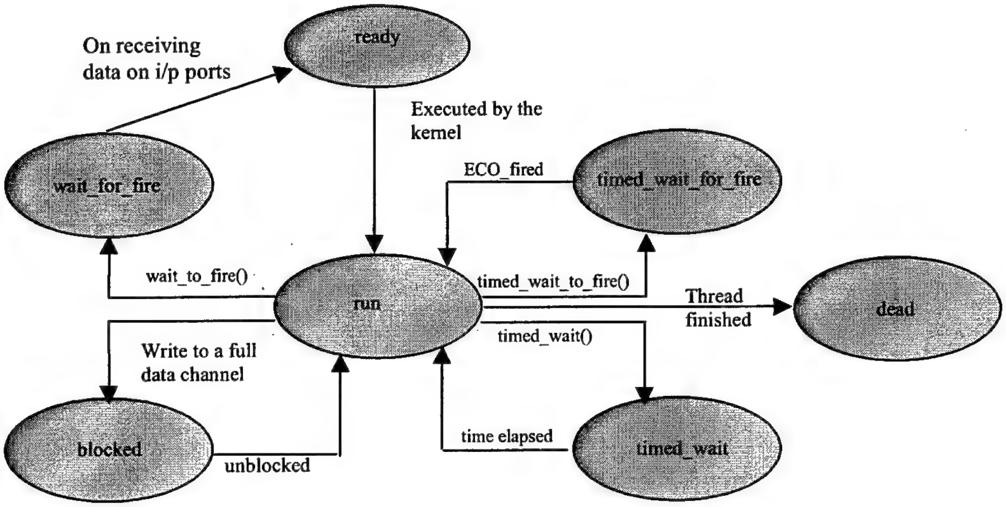


Fig. 3-32 Thread state diagram

The `wait_to_fire` function can be used to fire the ECO again. If the ECO is not ready for firing, it goes into the *wait_for_fire* state. The user can also delay the execution of the ECO for a pre-determined time, which puts the ECO into *timed_wait* state. The *timed_wait_for_fire* state is a combination of *wait_for_fire* and *timed_wait*. An ECO in this state can be fired if a firing mask becomes true *or* if the time period elapses. The ECO goes into the *dead* state once it finishes execution.

3.5.2.3.2 Context Switching

Operating systems can have one of two types of schedulers:

- Active Scheduler

- Passive Scheduler

An active scheduler runs as a separate thread and therefore necessitates a context save and restore (of the status of all the registers) every time there is a transfer of control between the scheduler and a process. A passive scheduler, on the other hand, does not run as a separate thread and is called by the process threads (through normal function calls). Although the passive scheduler approach obviates the need for explicit context save and restore, thus making it faster, this approach does not allow for preemption because if there has been a transfer of control from a process to the scheduler through a function call, the scheduler cannot suspend the currently running thread if need be, to run a new higher priority process. Control simply has to go back to the process thread from the scheduler. DARK++ therefore uses the active scheduler approach, in which context switches are brought about by the use of calls to the `setjmp` and `longjmp` functions. In the normal `setjmp` and `longjmp` calls, the context is entirely saved and restored, respectively. This means the contents of all the registers in the processor are saved during a `setjmp` and restored during a `longjmp`. However, DARK++ exploits the *dual-register-set* hardware provided by the Analog Devices SHARC 21160 micorprocessor for a substantially more efficient context switching. This approach is detailed in the following subsection.

Many digital signal processors used in embedded control systems, ADSP 21160 being one, have two sets of registers for increased performance – the *primary* set and the *alternate* set. DARK++ uses the primary register set for the kernel and the alternate register set for the process threads. Due to the use of two independent sets of registers for the kernel and the process threads, all that is required during a transfer of control between the two is flipping of a bit in a control register, which denotes the current *mode* (indicating whether the currently running thread uses primary set/secondary set), and saving/restoring some key status registers. DARK++ uses customized *setjmp* and *longjmp* assembly language procedures that selectively save/restore just these required registers.

Since most context switches in dataflow applications occur between the scheduler and executing threads, minimizing the cost of such switches increases the performance significantly. The use of the dual-register-set architecture in DARK++ for high-speed context switching between the scheduler and application threads has been found to reduce the switching time by 80% [vi].

3.5.2.3.3 Time Management

DARK++ provides APIs to allow ECOs to request a timed delay. In most other RTOSes, the kernel checks each waiting thread at every clock tick, and adds it to the ready queue when the waiting period has expired. However, this technique can introduce unnecessary overhead if there are a number of waiting threads. Hence DARK++ uses a different approach to handle timed delays. When the `timed_wait()` or `timed_wait_for_fire()` method is called, the delay is converted into an absolute time by adding the current system time to it and then stored in the ECO (process) object. The thread is then added to the waiting queue in which the threads are arranged in ascending order by absolute time and `actions_pending` is set to `future_actions`.

The kernel checks for `actions_pending` and adds the process back to the ready queue when the deadline has expired. To check whether the deadline has been reached, it compares the system time with the thread wakeup time of the first thread in the waiting queue. The scheduler needs to check only the first thread in the waiting queue unless that thread's waiting period has elapsed.

3.5.2.3.4 Interrupt Handling

There are many RTOSes that support interrupt handling through the use of compiler-provided mechanisms, using C functions that can be used as interrupt routines. This method involves a substantial overhead in context switching, since all registers are saved and restored while handling interrupts. The C compiler provided by Analog Devices for its SHARC DSPs supports this approach, and in addition, also provides the option of using the alternate register set for interrupt handling (since the C runtime uses only the primary register set). However, DARK++ cannot use this option, since it uses both the alternate and primary register sets.

DARK++ uses an alternative approach for handling external interrupts. This method provides performance comparable to that of using the alternate register set for interrupt handling. Here, rather than placing actions directly in the interrupt handler itself, DARK++ uses a minimal footprint handler that simply logs incoming events into the *event queue*, which is managed by the DARK++ scheduler. The interrupt handler runs in the currently active register set and only needs to save and restore a couple of registers. It logs a 32-bit code representing the interrupt that was received, into the event queue (a circular buffer of incoming events) and then returns control to the kernel. The status of the event queue is reflected by the `actions_pending` variable that we have already explained.

DARK++ also supports clock interrupts and non-maskable interrupts (NMI). The clock interrupt ISR is written in assembly and simply increments the kernel data member, `current_time` that is used for time management. Only a few registers required for incrementing a variable are saved and restored in this ISR. NMI is used for emergency condition notification and requires a time critical response. In most cases, it results in a call to the application's emergency shutdown procedure, bypassing all other kernel as well as application code.

3.5.2.3.5 Mutual Exclusion

Since threads have no shared memory and communicate only through data channels, most mutual exclusion problems do not arise in DARK++. However, there is one condition that needs to be handled. If interrupts occur when a process is executing, then control goes to the kernel. Under such a circumstance, it is important to ensure that the kernel resumes and completes the execution of the process thread that was suspended due to the interrupt, as soon as it executes the interrupt handler. This is because the event associated with the interrupt might have caused a higher priority process to be triggered. If this higher priority process is allowed to preempt the suspended process, then there is a possibility for data to be corrupted if the two processes access a common data channel. In such a scenario, however, the interrupt handler writes a 32-bit code into the event queue and control returns to the interrupted thread so that the event associated with the interrupt actually gets executed only after control returns to the kernel thread.

3.5.2.3.6 Volatile Declarations

In the embedded system context, it is imperative to have an efficient and optimized executable. With the optimizer enabled, typically several variables are cached in registers to make data fetches more efficient.

DARK++ uses the dual-register-set hardware for efficient context switching. This means that the kernel and the process threads work with distinct registers. Consequently, the two threads use different registers to cache the same data and this could lead to inconsistencies. It should be noted here that if all methods that manipulate data members private to their class were called through normal method call, then this problem does not arise, as these data members would not be cached in their callers. However, in the interest of our high performance objective, methods are all inlined, and this causes even the private data to be cached in registers, as part of the caller's thread. Therefore it is critical to identify all data that can potentially be accessed by both the kernel and the processes threads, and declare them *volatile* to ensure that they always get accessed from the memory instead of from registers.

The `front` and `rear` data members of the `Data_Channel` class, and the `in_ports_ready`, `wakeup_call` and `process_state` data members of the ECO class are declared volatile since these are accessed in the *Read/Write* operations, which can potentially be called from the kernel thread and the process thread.

In the context of interrupt service routines (ISRs), an important fact to consider is that interrupts could occur at anytime during the execution of the kernel/process thread and so if the ISR shares any data with either of these threads, then such data has to be declaredvolatile in order to avoid the executing thread from using an incorrect value that was cached in a register prior to the occurrence of the interrupt, after control returns back from the ISR. The `actions_pending` variable is declared volatile for this reason.

3.5.2.3.7 DARK++ Configurable Options

The DARK++ kernel can be configured to yield four distinct versions. These are obtained by selectively retaining/removing certain kernel features. Removal of features leads to an increase in performance with a concomitant reduction in run-time flexibility. The application designer can select the most appropriate DARK++ version for a given application's requirements. These are compile-time configurable by preprocessor macros. Table 1 lists the features in different versions.

The *full-featured* version of DARK++ has nothing disabled, and is a multi-threaded preemptive kernel. This version of the kernel schedules threads dynamically based on their firing rules and priorities. After every OS call (*read* and *write* operations), the scheduler is invoked to check for higher- and equal-priority threads. A context- switch takes place if and only if a higher- or equal- priority thread is ready. Preemption by an equal-priority thread ensures fair scheduling. This version is enabled by the preprocessor directive `PREEMPTIVE_MTHREADED`. It should be noted here that DARK++ offers preemption in a restricted sense, rather than the more common "textbook" sense, wherein preemption means stopping an executing thread as soon as a higher priority thread becomes ready. Here preemption happens only during API calls. However, the dataflow paradigm rules out the possibility of a higher priority process becoming ready while the current process is executing. This is because a process can become ready only when its source process has written data to its input data channel, but the source process thread could not have been running in conjunction with the currently executing thread.

The *non-preemptive* version of DARK++ does not invoke the scheduler on every OS call but instead, runs each thread until the thread suspends itself waiting for input data. Thus we avoid the overhead of calling the scheduler after every *read* and *write* operation by sacrificing immediate response to higher-priority threads. This version is enabled by the preprocessor directive NONPREEMPTIVE_MTHREADED.

The other two versions of DARK++ are single-threaded and avoid the time spent in context switching, thereby giving a significant performance boost to the application. The single-threaded approach is ideal for monotonic applications that execute sequentially, but unsuitable for applications that are highly dynamic. While the dynamically scheduled single-threaded version of DARK++ uses firing rules and priorities to select a process for execution, the statically scheduled single-threaded version uses a pre-computed firing order for threads, eliminating all use of priorities and firing rules, and is therefore the fastest. The single-threaded dynamically scheduled version is enabled by the preprocessor directive SINGLETHREAD_DYNSCHD and the single-threaded statically scheduled version is selected by SINGLETHREAD_STATSCHD.

These directives in turn use three other directives to control the features of the kernel – enable/disable preemption, enable/disable multithreading, and enable/disable dynamic scheduling. These are PREEMPTIVE, MTHREADED and DYNSCHD respectively. As mentioned, data Channels are of two types – message queues and mailboxes. A mailbox is an inter-processor data channel and is nothing but a special case of message queue where the size of the queue is one. This obviates the need for queue arithmetic and is therefore more efficient.

Multithreaded/ Single-threaded	Kernel Version	Preemptive/ Non-preemptive	Dynamically Scheduled/ Statically Scheduled
MTHREADED=1	**PREEMPTIVE_MTHREADED**	**PREEMPTIVE=1**	**DYNSCHD=1**
	NONPREEMPTIVE_MTHREADED	**PREEMPTIVE=0**	**DYNSCHD=1**
MTHREADED=0	**SINGLETHREAD_DYNSCHD**	**PREEMPTIVE=0**	**DYNSCHD=1**
	SINGLETHREAD_STATSCHD	**PREEMPTIVE=0**	**DYNSCHD=0**

Table 3-14 Configurable options in DARK++.

3.5.2.4 Real-time Support

Real-time scheduling algorithms could be based on fixed priorities or dynamic. While the rate monotonic priority assignment (RMA) algorithm, which assigns higher priority to shorter tasks is the optimal fixed-priority algorithm, the deadline driven scheduling algorithm is the optimal dynamic scheduling algorithm. The dynamic scheduling algorithms, however, in general, have a lot of overhead associated with them.

DARK++ provides the user the option of enabling real-time support. It has provision for the user to assign a function handle that will be used to run the provided scheduling algorithm. If this handle is null then the default algorithm that DARK++ uses is the fixed-priority RMA algorithm. In order to meet the high-performance objective, the complex real-time support necessary for POSIX compliance has been avoided in DARK++.

DARK++ provides the following simple API to monitor real-time deadlines.

A deadline for an ECO can be set using the method:

```
void ECO :: set_deadline(int time);
```

The `time` parameter specifies the time by which the ECO has to finish its execution. When this method is invoked, `time` is converted to an absolute time by adding the current system time to it and the ECO is added to the *deadline queue*.

The following method can be used to ascertain whether an ECO has met its deadline:

```
bool ECO :: check_deadline();
```

This method removes the ECO from the deadline queue and returns true if the ECO met its deadline.

The DARK++ scheduler checks the first entry of the deadline queue in each switching cycle. If it finds an ECO that missed its deadline, it calls a user-provided handler.

3.5.3 Experimental Evaluation

This chapter explains the performance experiments carried out with DARK++. It presents the performance results and a comparison with the corresponding results for DARK. We also explain the reasons and the implications of the obtained results. We have carried out empirical evaluation using three power control applications – the open-loop 3-phase inverter, the closed-loop 3-phase inverter and the closed-loop control for 3-phase boost rectifier.

In the following section we explain dataflow applications and the three applications used in our evaluation in particular, providing a brief overview of the various ECOs in each of them and an explanation of the primary functions performed in a switching cycle. We present the dataflow graph for each of them. Following this, in section 6.2 we present the results of the experiments and we conclude with a discussion of the obtained results

The following subsections explain three power-electronic dataflow applications that have been used to carry out performance experiments. These are – open-loop 3-phase inverter, closed-loop 3-phase inverter and boost rectifier. Following these, Section 6.2 presents the empirical evaluation details and the results.

3.5.3.1 Performance Results

This section discusses the results obtained during the performance evaluation experiments of the kernel. The experiments were conducted on Analog Devices-SHARC 21160 digital signal processor. The Analog Devices VisualDSP++ simulator was used to run the experiments and collect profiling information. Experiments were conducted on the three dataflow applications described.

We present the results obtained by comparing the performance of the different versions of DARK++ and DARK (full-featured, non-preemptive, single-threaded statically scheduled, single-threaded dynamically scheduled) on these three control applications. Table 3-15 through Table 3-17 show the data for the three applications run using message queues. Table 3-18 though Table 3-20 show the data for the applications when run using mailboxes. The tables show the total number of instruction cycles taken by DARK and DARK ++ for one switching period of the kernel for the mentioned applications. This is broken down into six categories of operations– ECO execution, dispatcher, context switching, ready queue operations and other operations. We can compare the contributions of each of these factors to the execution time for the application in the case of DARK++ with those in the case of DARK. Following this, we present graphs that are obtained by normalizing the overhead of DARK++ over the three control applications. Fig. 3-33 is based on the performance of the applications using queued data channels and Fig. 3-34 shows the same data for the applications run with mailboxes.

Table 3-15 Performance Results in terms of number of instruction cycles for the open-loop inverter– with message queue data channels.

Operations	Full-featured		Non-preemptive		Single-threaded dynamic scheduled		Single-threaded static scheduled	
	DARK	DARK++	DARK	DARK++	DARK	DARK++	DARK	DARK++
ECO execution	235	186	235	186	235	186	235	186
Dispatcher	529	425	529	425	219	177	212	192
Context switching	1148	1148	1148	1148	0	0	0	0
Ready queue operations	525	924	525	882	525	831	0	0
Data channel operations	1304	917	999	680	945	680	312	536
Other OS operations	462	626	460	486	77	280	0	0
Total	**4203**	**4226**	**3896**	**3807**	**2001**	**2154**	**759**	**914**

Table 3-16 .Performance Results in terms of number of instruction cycles for the closed-loop inverter– with message queue data channels.

Operations	Full-featured		Non-preemptive		Single-threaded dynamic scheduled		Single-threaded static scheduled	
	DARK	DARK++	DARK	DARK++	DARK	DARK++	DARK	DARK++
ECO execution	583	620	583	623	583	623	583	623
Dispatcher	679	553	679	553	297	270	270	268
Context switching	1476	1476	1476	1476	0	0	0	0
Ready queue operations	612	1719	612	1236	612	1031	0	0
Data channel operations	2659	2226	2059	1526	2059	1526	731	926
Other OS operations	594	881	594	727	203	467	0	0
Total	**6603**	**7475**	**6003**	**6141**	**3754**	**3917**	**1584**	**1817**

The execution time for the three applications, as we may note from the above tables, increases as the applications increase in complexity, with more number of ECOs and therefore, more computation, more communications, and also increased context switching and scheduling overhead. We see that the full-featured version involves maximum execution time since it provides the maximum number of features; it performs a check for an equal or higher priority ready process at the end of every *Read* and every *Write* operation and transfers control to the ker-

Table 3-17 Performance Results in terms of number of instruction cycles for the boost rectifier-- with message queue data channels.

Operations	Full-featured		Non-preemptive		Single-threaded dynamic scheduled		Single-threaded static scheduled	
	DARK	DARK++	DARK	DARK++	DARK	DARK++	DARK	DARK++
ECO execution	778	638	778	638	778	638	778	638
Dispatcher	1354	1057	1354	1057	585	495	567	529
Context switching	2952	2952	2952	2952	0	0	0	0
Ready queue operations	1811	2240	1811	2190	1811	2028	0	0
Data channel operations	4252	3762	3174	2439	3174	2439	1020	1852
Other OS operations	1224	2180	1224	1844	205	1317	0	0
Total	**12371**	**12829**	**11293**	**11120**	**6553**	**6917**	**2365**	**3019**

Table 3-18 Performance Results in terms of number of instruction cycles for the open-loop inverter – with mailbox data channels.

Operations	Full-featured		Non-preemptive		Single-threaded dynamic scheduled		Single-threaded static scheduled	
	DARK	DARK++	DARK	DARK++	DARK	DARK++	DARK	DARK++
ECO execution	235	186	235	186	235	186	235	186
Dispatcher	529	425	529	425	219	177	212	192
Context switching	1148	1148	1148	1148	0	0	0	0
Ready queue operations	525	924	525	882	525	831	0	0
Data channel operations	1040	521	734	307	734	307	67	280
Other OS operations	462	626	450	486	121	357	0	0
Total	**3939**	**3830**	**3621**	**3434**	**1834**	**1858**	**514**	**658**

Table 3-19 Performance Results in terms of number of instruction cycles for the closed-loop inverter– with mailbox data channels.

Operations	Full-featured		Non-preemptive		Single-threaded dynamic scheduled		Single-threaded static scheduled	
	DARK	DARK++	DARK	DARK++	DARK	DARK++	DARK	DARK++
ECO execution	778	638	778	638	778	638	778	638
Dispatcher	1354	1057	1354	1057	585	495	567	529
Context switching	2952	2952	2952	2952	0	0	0	0
Ready queue operations	1811	2240	1811	2190	1811	2028	0	0
Data channel operations	3699	2071	2640	1588	2640	1575	216	930
Other OS operations	1224	2180	1224	1844	88	1317	0	0
Total	**11818**	**11138**	**10759**	**10269**	**5902**	**6053**	**1561**	**2097**

Table 3-20 Performance Results in terms of number of instruction cycles for the boost rectifier – with mailbox data channels.

Operations	Full-featured		Non-preemptive		Single-threaded dynamic scheduled		Single-threaded static scheduled	
	DARK	DARK++	DARK	DARK++	DARK	DARK++	DARK	DARK++
ECO execution	583	620	583	623	583	623	583	623
Dispatcher	679	553	679	553	297	270	270	268
Context switching	1476	1476	1476	1476	0	0	0	0
Ready queue operations	612	1719	612	1236	612	1031	0	0
Data channel operations	2271	1041	1698	887	1698	888	220	333
Other OS operations	594	881	594	727	181	467	0	0
Total	**6215**	**6290**	**5642**	**5502**	**3371**	**3279**	**1073**	**1224**

nel if there is one. With the non-preemptive version of the kernel, an executing thread necessarily has to run to completion before another thread can begin execution, even if a higher priority thread becomes ready during the execution of the current thread. Hence process threads need to do no checking and switching of control to the scheduler. This significantly brings down the execution time. The single-threaded versions have no notion of separate process and scheduler threads. Instead, every process is run by a normal method call. Therefore single-threaded systems are necessarily non-preemptive. The single-threaded dynamically scheduled version supports the notion of firing rules and processes are scheduled dynamically based on the sequence of *Writes* and the corresponding triggers to the sink processes. Since the process to be run at any time is determined dynamically, this version of the kernel still involves ready queue management operations. Hence although it is slower than the earlier two versions discussed, it is slower than the single-threaded version in which processes have a pre-assigned execution order and the kernel essentially is a dispatcher and does no scheduling.

We now present the performance results for the three applications run with mailbox data channels. Mailboxes are data channels with unit capacity. Hence mailbox data channel management is much simpler involving no queue arithmetic, and simpler overflow handling. The above results indicate that DARK++ has performance comparable to that of DARK. The full-featured version of the kernel, in particular, outperforms that of DARK for the open-loop inverter and for the boost rectifier applications running on mailbox data channels, while the closed loop inverter running with mailbox data channels on DARK++ is a shade slower than on DARK (1.2% slower).

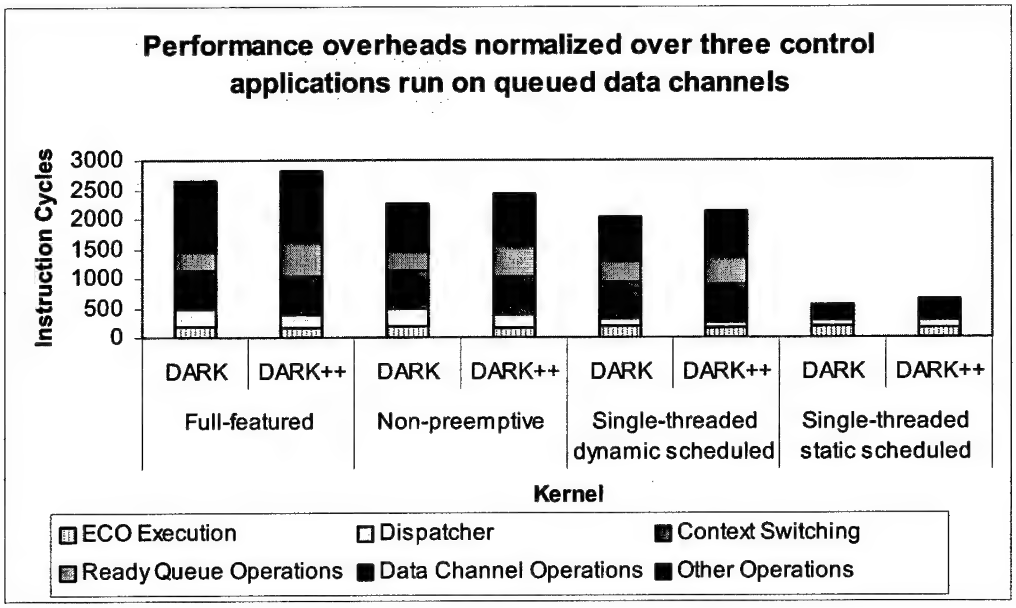


Fig. 3-33 Performance results for the two kernels with message queues.

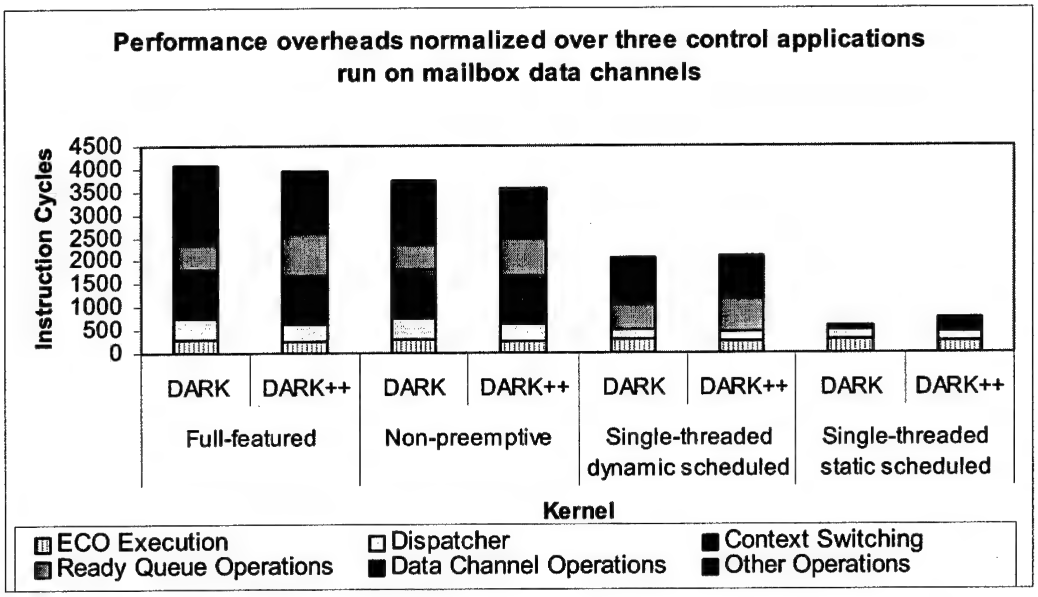


Fig. 3-34 Performance results for the two kernels with mailboxes.

Following are graphs that provide a good summary of all of the above data. They present the overheads imposed by each of the four versions of the two kernels, normalized over the three control applications. The first graph is for the applications run using message queues and the second one is for the applications run using mailboxes. After presenting these graphs, we will discuss the results gathered by comparing the OO versus the non-OO kernel.

3.5.3.2 Discussion of the performance data

It may be observed that the multithreaded versions of DARK++ running the applications using message queues resulted in marginally lesser speed than that of DARK, while running the applications using mailboxes showed a performance gain. The data channels operations, which are implemented as macros in the C version of the kernel, are inlined methods in the C++ versions. Because these operations were actually being inlined by the compiler more often in the case of mailboxes than in the case of message queues, there was a significant performance gain.

The single-threaded versions of DARK++ impose significant overhead. The DARK++ dispatcher for this version is slower than the DARK dispatcher unlike for the other versions. This is because, while the multithreaded kernel uses calls to *setjmp* to execute processes in all but the first switching cycle of the kernel, the single-threaded kernel always makes an explicit call to the ECO Implementation method and since this is a virtual method, there is a considerable overhead introduced due to the dynamic resolution to effect the call.

The context switching times are equal in DARK and DARK++ since both use the same custom *setjmp* and *longjmp* assembly functions to accomplish this. While the ECO execution and the scheduler execution times are comparable in the two cases, the ready queue operations have a significantly higher contribution to the entire execution time in the case of DARK++ than in DARK. This is because all accesses to any of the ready queue data members have been counted under this category and there are a number of calls to such operations – e.g., calls to a method that returns the number of items present in the ready queue. Such calls are made both by the scheduler, as well as from the data channel operations in the case of preemptive scheduling, to check for other high-priority ready processes that may be waiting. Typically, it has been observed that simple methods that return the value of a data member take between 7-9 instruction cycles. Hence even if such a method is inlined by the compiler, there is an overhead incurred by the frequent use of such methods.

The "other operations" category also takes more number of instruction cycles in the case of DARK++ than in DARK due to the same reason as mentioned above. There are some generic methods that are frequently used by various callers to retrieve some data members and these introduce significant overhead.

It is worth noting that a great many operations that are specified as macros in DARK are class methods in DARK++, with the "inline" keyword. Therefore, while these operations are guaranteed to be preprocessed and efficient in DARK, many of them are not inlined by the compiler in DARK++. A better performance could have been achieved with DARK++ if it were possible to guarantee inlining of all methods. Also, the unpredictability of a method actually being inlined could lead to marginal irregularities in performance.

The single-threaded versions of the kernel need do no context switching and hence the number of instruction cycles for the category is zero. The single-threaded statically scheduled kernel makes use of a precomputed order to execute the processes sequentially and hence does not use the ready queue.

3.5.3.3 Summary

From the data gathered on these three applications and from the above discussion, we may conclude that careful design in OO paradigm can yield appreciable performance. We summarize below, the most important points about OO design and performance issues:

- As we have seen, it very naturally imposes the need for more method calls. While one can choose to specify such methods with the "inline" keyword, since it relies on the discretion of the compiler, there may be inefficiencies (if the compiler does not actually inline them). The disadvantage with inlining is that for huge applications, the entire code may not fit into memory if the memory offered by the embedded system hardware is limited.

- Another related point is that, while it is often worthwhile to specify some frequently used (small) operations as macros in C, it may be inappropriate to do this in C++ (an OO language) where more often than not, we want operations as methods in a class and specifying these as macros might lead to a sloppy design. In DARK++, as stated earlier, we have specified a few generic operations used by the data channel *Read* and *Write* methods as macros. The question really is a tradeoff between elegance and performance.

- It is best to avoid virtual methods as these rely on dynamic binding, which impact performance considerably. The performance numbers for the single-threaded statically scheduled version of DARK++ reflect this fact very clearly. However, if the system being designed compels the use of virtual methods, one necessarily pays for the V-table look-up and resolution during run-time. However, we often lose the flexibility and natural extendibility through inheritance when we avoid usage of virtual methods.

- There are some important points to remember while working on OO design for performance-critical systems. Use of dynamic memory allocation, perhaps by creating objects "on the fly" is a bad idea for a system where performance is critical. This should be avoided.

- Templates are often handy and neat to use in the OO design and user-defined templates do not have any inherent performance concerns associated with them since template instantiations take place before run-time.

3.6 Transparent Distributed Messaging

The protocols designed so far for power electronics systems are for single controller systems or multiple controller systems with fixed processor allocation. Single controller systems fail to use the advantages offered by distributed systems, which are improved efficiency and greater fault tolerance. Using multiple controller systems with fixed processor allocation severely restricts the flexibility and hence the usage of the system.

In this report, we present a protocol for transparent inter-processor communication across a network thereby allowing transparent distribution of any multi-controller application. The protocol is designed such that it can run the same application without any kind of code change in virtually any kind of distributed configuration, where configuration is the number of controllers used in the system plus the processor allocation strategy used. The protocol works well even for single-controller applications and for pre-defined allocation of processors to controllers. The protocol, thus offers a lot of flexibility and ease of use in running a multi-controller application and evaluating its performance using different number of controllers and/or processor allocation strategy. The protocol also enables an application, with an automated processor allocation strategy, to transparently configure itself for any number of processor nodes without requiring any changes or recompilation.

Dataflow architecture is a software architecture used to design software for plug and play power electronics building blocks. It is a data-driven architecture consisting of a large number of program elements to support component-level design. The embedded system being designed consists of a number of Elementary Control Objects (ECOs), which are concurrently executing entities. The ECOs are connected to each other through data channels. The ECOs read data from the data channels, process the data and generate the output.

The ECOs are scheduled by their firing rules. Firing rule for an ECO indicates the input channels on which the ECO should wait for data before being fired. A read on a data channel can unblock an ECO waiting to write data into that data channel. Similarly, a write operation can fire an ECO waiting for input on that data channel. The execution of the ECOs is managed by the DARK Operating system [2].

The application programmer provides a DFG (Dataflow graphs) descriptor file, which contains information on the number of ECOs present in the system and the data channels that connect them. The application programmer also provides implementation of the ECOs.

The protocol makes use of the existing dataflow architecture to provide for transparent message passing between ECOs present on different controllers.

3.6.1 Design and implementation

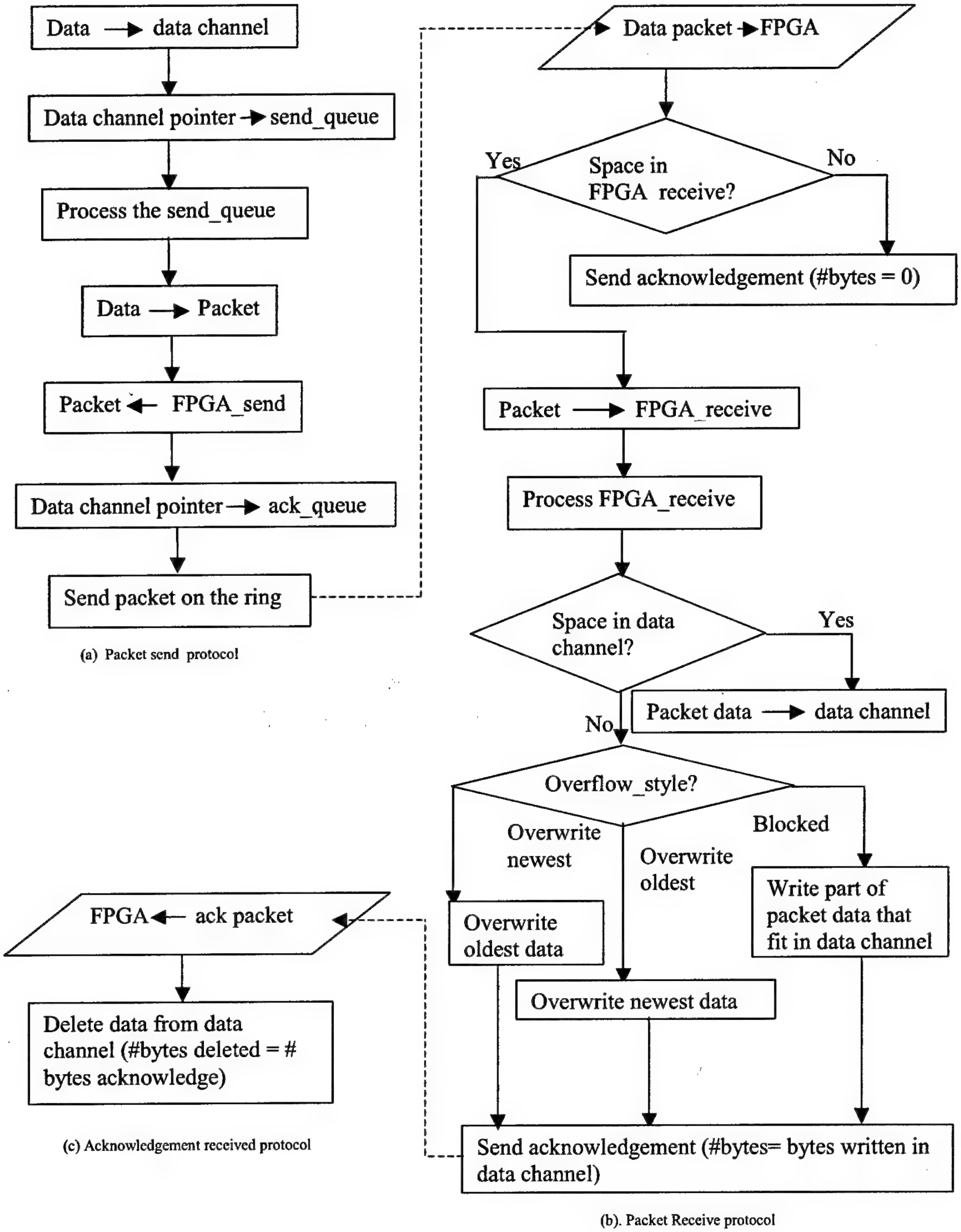
Based on the number of controllers available, the system automatically allocates ECOs to different controllers. The controllers will be placed on a ring, which operate based on a protocol called PESNET [x]. The ECOs communicate asynchronously with each other by reading or writing data into the data channel. They are not aware of the number of controllers present in the system and hence of the distributed nature of the communication.

The protocol needs to ensure that communication between ECOs on different controllers is carried out transparently. This necessitates orderly arrival of messages and special handling of loss messages.

3.6.1.1 Design

Fig. 3-35 (a) describes the protocol for sending a data packet from the source ECO to a destination ECO, where the ECOs are on different controllers. The ECOs on a single controller communicate with each other by reading or writing data from data channels. Distributed communication also occurs through data channels. In case of distributed communication, the source ECO writes data to a distributed data channel. The OS processes the data in the data channel, packs it into packet and passes the packet onto the FPGA. Note that the OS does not yet remove the data from the channel. The packet is then sent onto the ring by the FPGA.

Fig. 3-35 (b) describe the protocol on the receiver side. When the FPGA receives a data packet, it stores the packet, to be later processed by the OS. The OS extracts data from the data packet and writes it to the data channel identified by the packet. It then prepares an acknowledgement packet acknowledging the number of data bytes written to the data channel. The acknowledgement packet is then passed to the FPGA. The FPGA sends the packet over the ring. Fig. 3-35 (c) explains the protocol when an acknowledgement packet is received. The OS deletes the data from the data channel based on the number of bytes acknowledged.



(a) Packet send protocol

(c) Acknowledgement received protocol

(b). Packet Receive protocol

Fig. 3-35 Dataflow messaging protocols.

3.6.1.2 Data structure

The protocol makes use of two circular buffers – the send_queue and the ack_queue. A send_queue entry points to a distributed data channel that has data to send. An ack_queue entry points to a distributed data channel that is awaiting an acknowledgement for the packet sent. Note that the send_queue and the ack_queue together will contain not more than one entry corresponding to each distributed data channel. The size of the send_queue and ack_queue is equal to the number of distributed data channels in the system.

The status of a data channel is indicated by the value stored in the alloc_type field of the data channel. The alloc_type field of a distributed data channel can have one of the three values

1. WAITING_TO_SEND when the data channel contains data to be sent across the ring
2. SENT when a packet has been sent and an acknowledgement is awaited and
3. EMPTY when the data channel is empty and is not waiting for an acknowledgement.
4. The alloc_type field of a normal data channel will always have value NULL.

The FPGA uses two fixed size buffers - FPGA_send and FPGA_receive. The FPGA_send stores packets to be sent on the ring while the FPGA_receive stores packets received from the ring. Data structures for the protocol are as shown in Fig. 4. Note that the size of the send_queue and ack_queue is equal to the number of distributed data channels.

3.6.1.3 Implementation

The sender's side protocol is described by the dotted lines in Fig. 3-36. When an ECO writes data into a distributed data channel with alloc_type field as EMPTY, a pointer to the data channel is stored in a send_queue entry. The alloc_type field of the data channel is changed to WAITING_TO_SEND.

When the operating system scheduler is called, it checks for entries in the send_queue. If the send_queue is not empty and there is space in the FPGA_send buffer, the scheduler reads data from the data channel pointed to by the send_queue entry writes it into a packet and then stores the packet in the FPGA_send buffer. A pointer to the data channel is removed from the send_queue and added in the ack_queue. The alloc_type field of data channel is changed to SENT. Note that the data is not yet removed from the data channel.

Fig. 3-37 shows the packet structure. The packet contains addresses of the source and destination controllers. The datachannel_id field is used to uniquely identify the data channel and thereby the ECOs associated with the data channel.

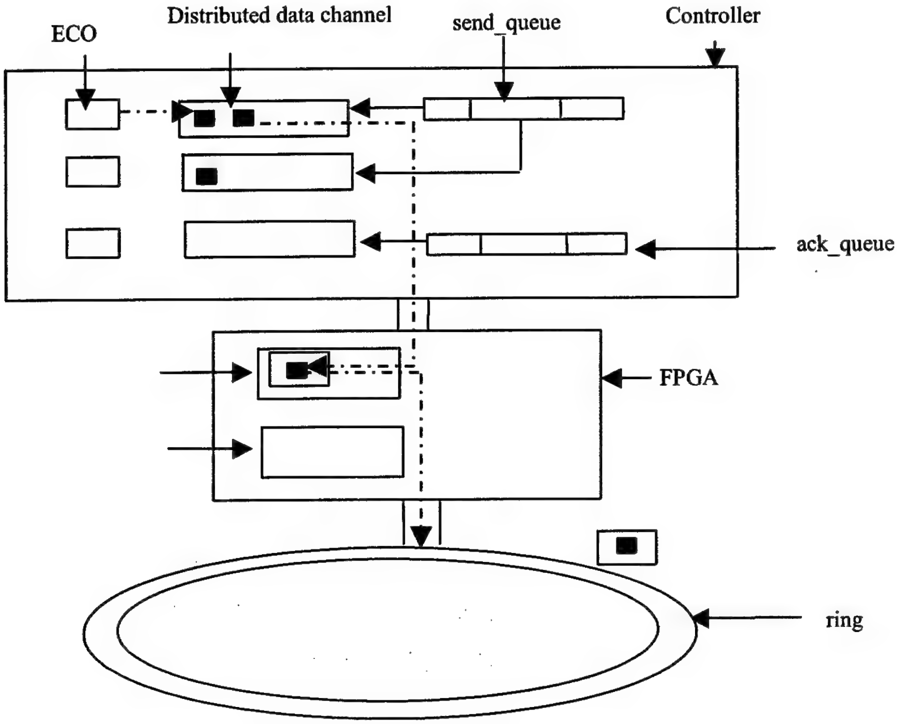


Fig. 3-36 Data structures.

```
typedef struct
{
 Net_Address from_address   : 8;
 Net_Address to_address     : 8;
 char datachannel_id        : 16;
 packet_command command     : 4;
 unsigned int number_of_bytes: 4;
 char data [9];
} Digested_Packet;
```

Fig. 3-37 Packet structure.

The command field is use to indicate the packet type. The command field can have one of the two values -

1. data_packet
2. ack_packet

For a data packet, the number_of_bytes field indicates the number of data bytes contained in the packet. The number_of_bytes field for an acknowledgement packet indicates the number of data bytes acknowledged by the receiver. 4 bits have been allocated for the command field to provide for future improvements.

When the FPGA gets an empty token on the ring, it grabs the token and passes the data packet onto the ring.

The FPGA on the receiver side removes the packet from the ring and stores it in the FPGA_receive buffer, if there is space in the buffer. If the FPGA_receive buffer is full, the FPGA sends out an acknowledgement packet acknowledging zero bytes of data.

The operating system scheduler checks the FPGA_receive buffer for any incoming packets. If the receive buffer is not empty, it reads the packet and checks if there is space in the data channel identified by the packet. If there is space for the entire data packet, then the data packet is written into the data channel. Otherwise, the scheduler overwrites the oldest data element or the newest data element or writes part of the data that fit into the available space in the data channel. These actions are based on value of the Overlflow_style field of the data channel. In all cases, an acknowledgement packet is sent back acknowledging the number of bytes written into the data channel.

When an acknowledgement packet is received, the number of bytes acknowledged determines the number of bytes to be deleted from the data channel. The pointer to the data channel in the ack_queue is then removed.

3.6.1.4 Fault Tolerance

In order for the messaging to be transparent, the design needs to ensure that packets arrive in order. Orderly arrival of packets is ensured by requiring every packet to be acknowledged and the next packet for a given data channel be sent only after an acknowledgement is received for the previous packet.

When a packet is sent over the network, a copy of the data is stored on the sender's side and is deleted only after an acknowledgement for the data is received. If there is not enough space for the entire packet data in the data channel on the receiver side, it is possible that only part of the data gets written into the data channel and hence only part of the data gets acknowledged. In that case, only those data bytes that are acknowledged are deleted and an attempt is made to re-send the unacknowledged data bytes.

For packets lost due to node or ring failure, the protocol relies on the underlying PESNet protocol to ensure fault tolerance. The PESNet protocol makes use of a dual, counter-rotating fiber optic rings to improve the fault tolerance of a network. In case of a node or a link failure, the bi-directional ring allows the message can backtrack. Thus, the PESNet protocol ensures that there won't be any missing packets irrespective of a node or a link failure.

3.6.2 Analytical performance assessment

An analysis of the system performance is performed based on factors such as network speed, number of nodes on the ring and saturation of the network.

Let us consider a network of N nodes. Let P be the size of a packet in bits and R be the transmission rate of the network. Then, the time required for a complete cycle of a packet that is the time between the transmission of a packet by a sender, its processing at the receiver and the return of an acknowledgement packet from the receiver back to the sender is given as

$T_{cycle} = T_{process} + T_{send} + T_{ack}$

where,

T_{cycle} = Time to transmit packet from sender to receiver

$T_{process}$ = Time to process the packet at the receiver

T_{ack} = Time to transmit the acknowledgement from the receiver back to the sender

The time to transmit a packet from sender to receiver depends on the time to send a packet over a single network link, the number of hops between sender and receiver and the saturation of the network. Hence,

$T_{send} = T_{sat\_delay} + \text{Distance x } T_{packet}$

where,

T_{packet} = time to send a packet over a single link,

T_{sat_delay} = Delay due to network saturation for a single packet and

Distance = # of hops between sender and receiver.

The network can be considered as divided into slots of data packets. A packet can be sent over the network only at the start of a packet slot. Since the time to process a packet at the receiver is very small, it can be safely assumed that the processing time is equal to length of a single packet slot and hence equal to the time to transmit a packet over a single link.

$T_{process} = T_{packet}$

The time to transmit an acknowledgement packet can be given as

$T_{ack} = T_{sat\_delay} + (N - \text{Distance}) \text{ x } T_{packet}$.

Hence, the total cycle time can be written as

$T_{cycle} = 2\text{x } T_{sat\_delay} + (N + 1) \text{ x } T_{packet}$

If s is the saturation index of the network with value between 0 and 1, then the saturation delay can be given as

$T_{sat\_delay} = 1 / 2 \text{ x } s / (1\text{-}s) \text{ x } T_{packet}$.

If T_{delay} is the delay introduced by each node in the network, then

$T_{packet} = P / R + T_{delay}$.

Hence, the time for a complete cycle can be given as

$\mathbf{T_{cycle} = (N + 1 + s/(1\text{-}s)) \text{ x } T_{packet}}$.

To provide a basis for concrete discussion, we consider an example application with two controllers and 6 phase legs. The controllers are switching at a frequency of 20KHz. As there are 2 controllers and 6 phase legs, the number of nodes, N in the network is 8. Let us assume that the network saturation is 25% that is value of s is 0.25. As 8 data bits get transmitted as 10 bits due to 4B/5B encoding by the transceivers, a packet of size 16 bytes will give a value of P as 160 bits. If the network speed R is 100Mbytes per second, and the delay introduce by each node, T_{delay} is 3 nanoseconds, then the total cycle time will be around 15 microseconds.

Thus, it will be possible to perform 3 such cycles in a single switching period of 50 microseconds.

3.7 Comparisons between the dataflow approach, Matlab Simulink, and Real-time Workshop package

Several commercial software packages have been developed to provide graphical control software design and simulating environment. For example, Mathworks Simulink [xi] is a widely used software package for modeling, simulating and analyzing dynamical systems; and Real-time

Workshop [xii] generates optimized, portable and customizable code from Simulink models, which could run on many production targets. These two software packages together provide a software platform for rapid prototyping process and automatic program building.

From software construction point of view, both approaches attempt to reduce engineering effort by construct software from standard functional blocks in design libraries. Dataflow architecture allows users to design software from functional self-contained library blocks at the C code level. Simulink and Real-Time Workshop save software design effort by providing a graphical modeling environment and automatic C code generation. Though dataflow approach requires a dataflow graph description, which is handwritten so far, this software architecture has the potential to incorporate a graphical design interface to further reduce the software design period and cost.

However, the two approaches differ dramatically at the constructed software. A real-time kernel designed for dataflow architectural software provides abundant real-time control features to meet requirements from different kinds of applications. These real-time control features range from static single thread scheduling to preemptive multithread scheduling. The bare board embedded C code generated from Simulink and Real-Time workshop supports only single tasking or preemptive multitasking. Also because of the natures of infrastructures of the generated software, the dataflow architectural software is easy to design for distributed control application, while Simulink and Real-Time Workshop is more suitable for centralized control software design.

3.7.1 Overview of Mathworks Simulink and Real-Time Workshop Software

In recent years, MathWorks Simulink and Real-Time Workshop software packages have been widely used in industry and academia for modeling and simulating dynamic systems and generating C code for rapid prototyping or embedded control. Simulink is a software package for modeling and simulating dynamic systems. It provides a graphical design environment that allows designers to build models as block diagrams. Real-Time Workshop generates optimized, portable and customizable ANSI C code from Simulink models to create stand-alone implementations of models that operate in real-time and non-real-time in a variety of target environments. The relationships between Mathworks' MATLAB, Simulink and Real-Time Workshop are shown in Fig. 3-38.

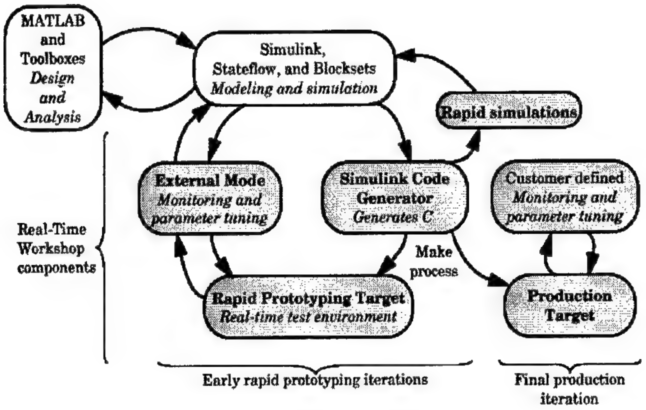

MATLAB and Toolboxes *Design and Analysis*
Simulink, Stateflow, and Blocksets *Modeling and simulation*
Rapid simulations
External Mode *Monitoring and parameter tuning*
Simulink Code Generator *Generates C*
Customer defined *Monitoring and parameter tuning*
Real-Time Workshop components
Make process
Rapid Prototyping Target *Real-time test environment*
Production Target
Early rapid prototyping iterations
Final production iteration

Fig. 3-38 Relationships between MathWorks MATLAB, Simulink and Real-time Workshop software packages.

The overall software design procedure using Simulink and Real-Time Workshop package is first draw block diagram based system model in Simulink, and then build target C code using Real-Time Workshop. Simulink provides standard block libraries and a graphical design environment. It supports hierarchical design, which means a block can be composed of several sub blocks. It also allows users to create their customized library to simplify their specific design procedure. The designed model can be simulated in Simulink to adjust system model structure or model parameters. After several such iterations, when the simulation results match the design specifications, the model can be translated into C code through Real-Time Workshop. Real-Time Workshop allows users to choose from several code formats for different code running targets. To generate C code that can be complied for embedded systems, Embedded Coder format is applied for the design example in this paper.

The overall software design procedure using Simulink and Real-Time Workshop package is first draw block diagram based system model in Simulink, and then build target C code using Real-Time Workshop. Simulink provides standard block libraries and a graphical design environment. It supports hierarchical design, which means a block can be composed of several sub blocks. It also allows users to create their customized library to simplify their specific design procedure. The designed model can be simulated in Simulink to adjust system model structure or model parameters. After several such iterations, when the simulation results match the design specifications, the model can be translated into C code through Real-Time Workshop. Real-Time Workshop allows users to choose from several code formats for different code running targets. To generate C code that can be complied for embedded systems, Embedded Coder format is applied for the design example in this paper.

To generate embedded C code, there are some constrains on model blocks. It requires that all blocks in the model are either discrete time block or continuous time block but can be sampled at discrete time. If multiple sample rates are used in a system, it requires that the lowest sample will be chosen as the base rate and other higher sample must be multiple time of the base rate. The purpose of these constraints is for the Embedded Coder to generate C code with some basic real-time scheduling support.

The C code generated from Embedded Coder is in legacy main-program-and-subroutine style, composed of a sample main program, an interrupt service routine (ISR) to implement the control algorithm and data structure descriptions. The pseudo code of the main program and the ISR, rt_OneStep (), is shown in Fig. 3-39. In the main program, after initialization, the DSP enters an infinite loop to wait for interrupts. The interrupts occur at the base sample specified in the Simulink model. And in the ISR, ModelStep is called to implement control in the current time step. The structure of ModelStep is shown in Fig. 3-40, where MdlOutput computes the outputs of a model, MdlUpdate updates model states, and MdlDerivatives computes derivates for model states if necessary.

```
main()
{
 Initialization (including installation of rt_OneStep as  an interrupt service routine for a real-
time clock)
 Initialize and start timer hardware
 Enable interrupts
 While(not Error)and (time <final time)
   Background task
 EndWhile
 Disable interrupts (Disable rt_OneStep from executing)
 Complete any background tasks
 Shutdown
}
```

(a) Pseudo main program.

```
rt_OneStep()
{
        Check for interrupt overflow or other error
        Enable "rt_OneStep"(timer)interrupt
        ModelStep—Time step combines output, logging, update
}
```

(b) Pseudo ISR program.

Fig. 3-39 Pseudo code of Embedded Coder generated C program.

3.7.2 Comparison of Dataflow Approach and Simulink & Real-time Workshop Package

A 3-phase voltage source inverter (VSI) with closed voltage loop is chosen as the design example. The specifications of the 3-phase VSI are:

Input: Vdc = 200 V;

Outputs: balanced 3-phase sinusoidal with line-to-line voltage 100V;

Switching frequency: fs = 10kHz;

Output inductance L = 300 uH at each phase;

Output capacitance C = 100 uF at each phase.

The voltage loop is design to have phase margin 35 degree and 10 dB gain margin.

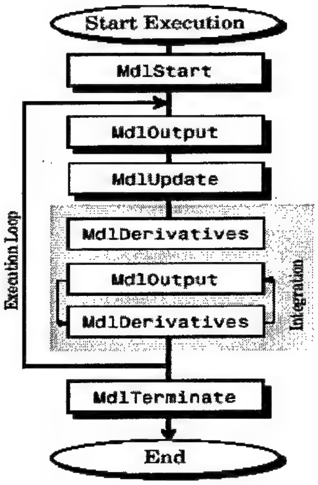


Fig. 3-40 ModelStep structure.

The dp transformation technology is used to simplify the close loop control design and SVM technology is used to implement the modulator. The digital controller is assumed to run in an Analog SHARC DSP (ADSP 21160). Analog Device also provides a software development environment Visual DSP/Visual DSP ++, which support ANSI C.

3.7.2.1 Software design procedure

Fig. 3-7 shows the dataflow graph of the close loop control of the 3-phase VSI, while Fig. 3-41 shows its Simulink mode. From the high end user point of view, the two software construction approaches have similarities. For the user of Simulink and Real-Time Workshop package, the main task is using Simulink as a graphical interface to drag and pull blocks from design libraries and then chooses a desired target for the Real-Time Workshop to compile into C code. When using the dataflow approach, the designer only needs to provide a dataflow description file to describe ECOs and their connections.

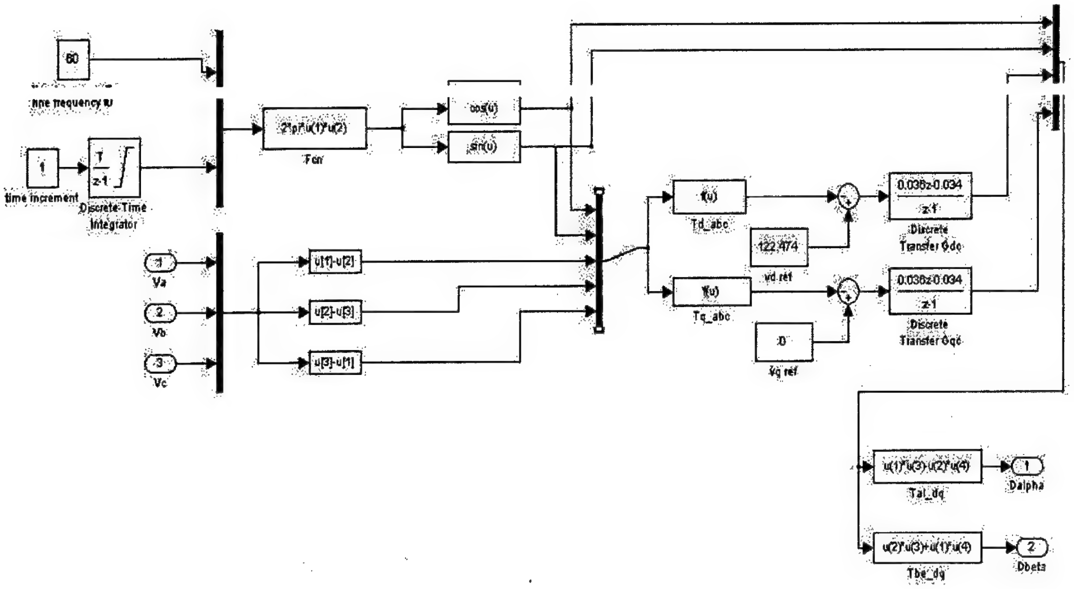


Fig. 3-41 Simulink Model of voltage close loop control of 3-phase inverter.

3.7.2.2 B. Code structure and performance analysis

Though there is a significant similarity in Fig. 3-7 and Fig. 3-41, both composed of function blocks and data connections, the generated code are in different structures because of different block implementation methods and block connection mechanisms.

The generated code from Real-Time Workshop Embedded Coder [xiii] is in main-program-and-subroutine style as presented in section III. A block in a Simulink model has two tasks during one time step: compute output and update states if necessary. In the generated C code, output computation for individual blocks are combined into MdlOutput, while states updating for individual blocks are combined into MdlUpdate. Inter-procedure calls are reduced in order to optimize the generated code for real-time execution. The block execution sequences inside MdlOutput and MdlUpdate are prescheduled. Arrows in Fig. 3-41 are translated into singular storage units and used to decide the prescheduled block execution sequence.

Blocks in Fig. 3-41 are independent processes in dataflow software, while arrows are data channels. Each process has its self-contained functionality. The execution sequence of processes can be statically scheduled, or dynamically scheduled. In dynamically scheduling, Each ECO process can be activated at its own sample rate and the activation depends on its input data channels status. There is no constraint between sample rates of different ECO process. However, context switching between ECO processes and maintaining data channels introduce run-time performance overhead.

Fig. 3-42 shows the DSP execution cycles during one switching period for Embedded Coder generated C code and dataflow software with different real-time kernel features. The code efficiency of Embedded Coder generated C code is compared to that of dataflow C code with mailbox data channels and static single thread scheduling. From Fig. 3-42, it can be seen that the Embedded Coder generated C code actually takes more time on computation than any dataflow counterparts. What the Embedded Coder optimized during its code generation is mainly reduced inter-procedure calls.

Since the Embedded Coder generated C code is in main-program-and-subroutine style, it has the drawbacks inherent in its software style. First, the generated code is naturally fit in centralized control structure. Significant extra engineering effort will be involved to split the generated code into distributed control system since the blocks in Simulink model are flattened. Second, there is possibility that the generated code is not absolutely compatible with the target compiler, which makes the software debug task tedious because the generated code contains MATLAB specific definitions. Small changes in the generated code may cause the designer goes back to the Simulink models.

For dataflow software, though it introduces run-time performance overhead, every ECO process is independent, which makes the software easily run in distributed control system or multi-processor system. The inter-process communications is carried through data channels, which can be designed upon network communication protocols and transparent to applications. The ECO processes allocation mechanisms have been designed and the ECO inter-processes communication protocol is under design [xiv].

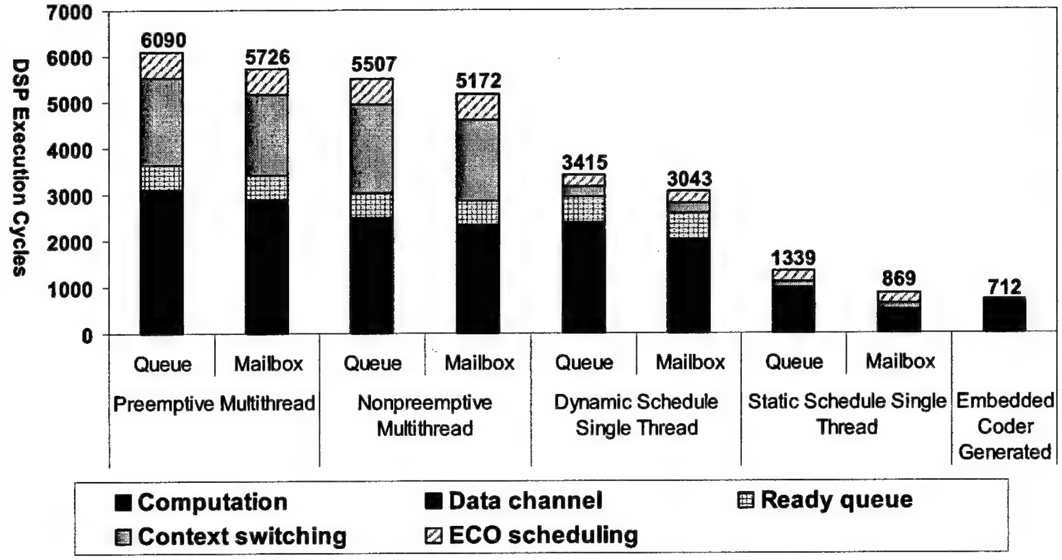


Fig. 3-42. Code performance comparison.

3.8 Conclusion

By focusing on software architecture, we proposed dataflow approach to designing power electronics control software that is built from standardized modules, possesses a higher degree of reusability, and supports greater reconfigurability. An appropriate architectural design for power conversion system has been constructed, which is a key element of its long-term success. We implemented quite number of control applications in dataflow architecture. We experimentally assessed the context switching time of the selected micro-kernel. We also experimentally measured the additional overhead imposed by the ECO-based dataflow style by comparing a small collection of control algorithm implementations against existing baseline versions written as traditional C programs. In order to adpat dataflow control software to distributed computation environment, we designed transparent messing protocols. Finally, we compared our approach to other commercially software platforms.

4 HARDWARE MANAGER

4.1 Introduction

At the core of the research done on the PEBB concept are the ideas of reliability, flexibility, and modularity. PEBB research is establishing power conversion systems that are distributed, reliable, and flexible in nature. The Universal Controller is the keystone to these principles; the remaining stones are the Hardware Managers. The Hardware Manager board interfaces the control loop to the power stage. On one side, the Hardware Manager connects into the information system via optical fibers arranged in single or dual ring structure. On the other side, the Hardware Manager connects to the power stage through a phase leg, forming a basic PEBB. The PEBBs are combined in several fashions to obtain a number of converter topologies (please see Chapter 5 – Power Stage). The idea is not to concentrate on a specific topology or power level, for the system is independent of such constraints (up to a limit). Thus we see the flexibility and modularity of the PEBBs. The following chapters describe the design, operation, testing, and the future research associated with the Hardware Manager.

During the past year of this project, a brand-new Hardware Manager board was designed, manufactured, assembled, and tested. Fig. 4-1 shows a picture of the top and bottom views of this board. The new Hardware Manager controls newly developed 33kW PEBB hard switched phase legs, shown in Fig. 4-2. The new Hardware Manager now uses a Xilinx FPGA, has support for a dual-ring communication protocol, built-in protection mechanisms, as well as several debugging features.

On a board level, the Hardware Manager was designed similarly to the PEBB. Every functional part of the board, from communications to the sensors, is basically independent of each other. The VHDL code in the Hardware Manager, just as in the Universal Controller, can be instantiated individually, and thus making troubleshooting much easier.

The communications on the new Hardware Manager has been shown to be more reliable, and is working just as expected. The operation of the sensors has also been verified. Some errors were found with this new version of the Hardware Manager, but they have been easily correctable, cosmetic mistakes. Overall the Hardware Manager board has been a big success, considering the development and troubleshooting time, and its potential for the future.

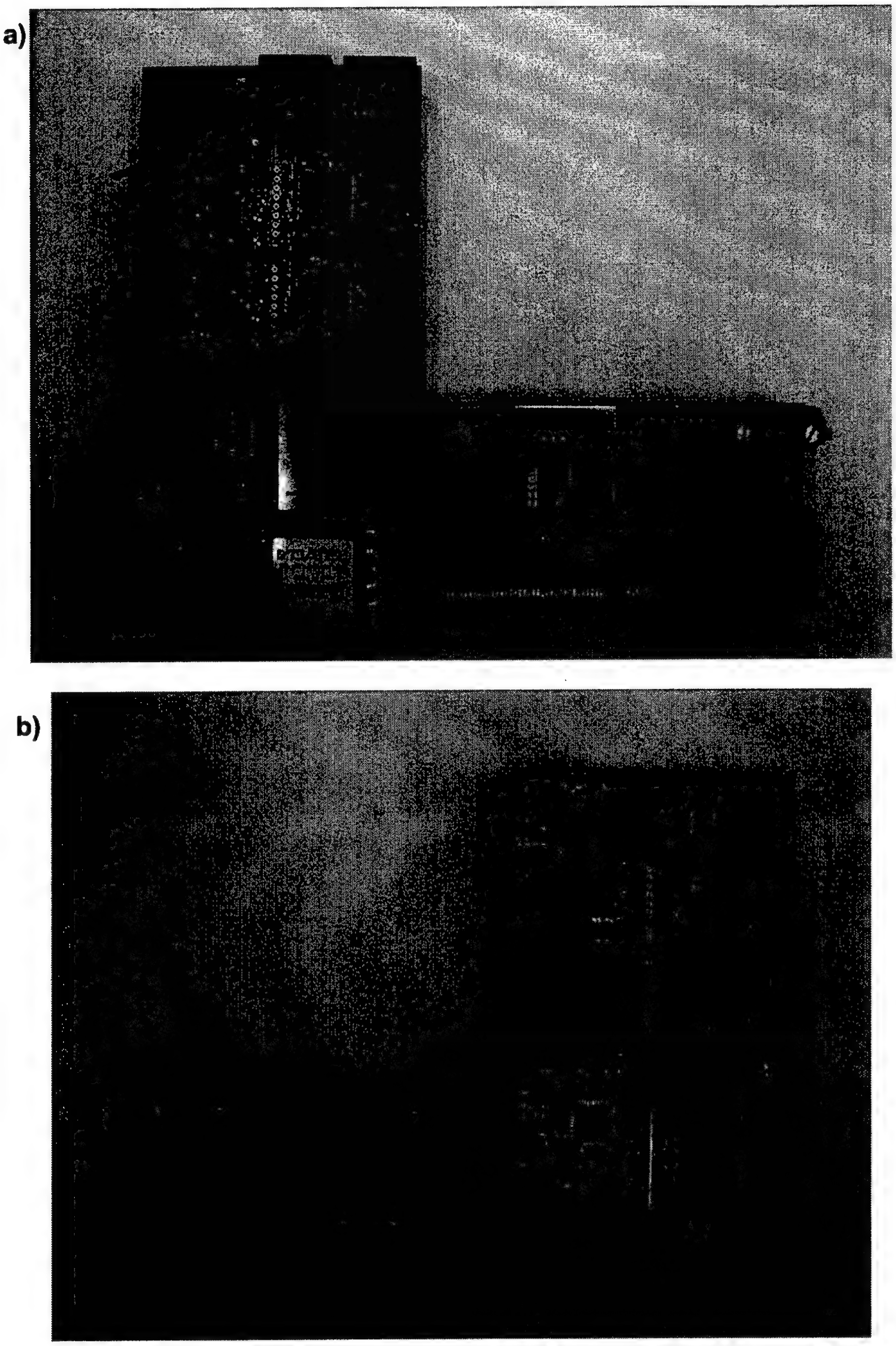


Fig. 4-1 New Hardware Manager developed for the PnP PEBB-based power electronics systems, a) top and b) bottom views.

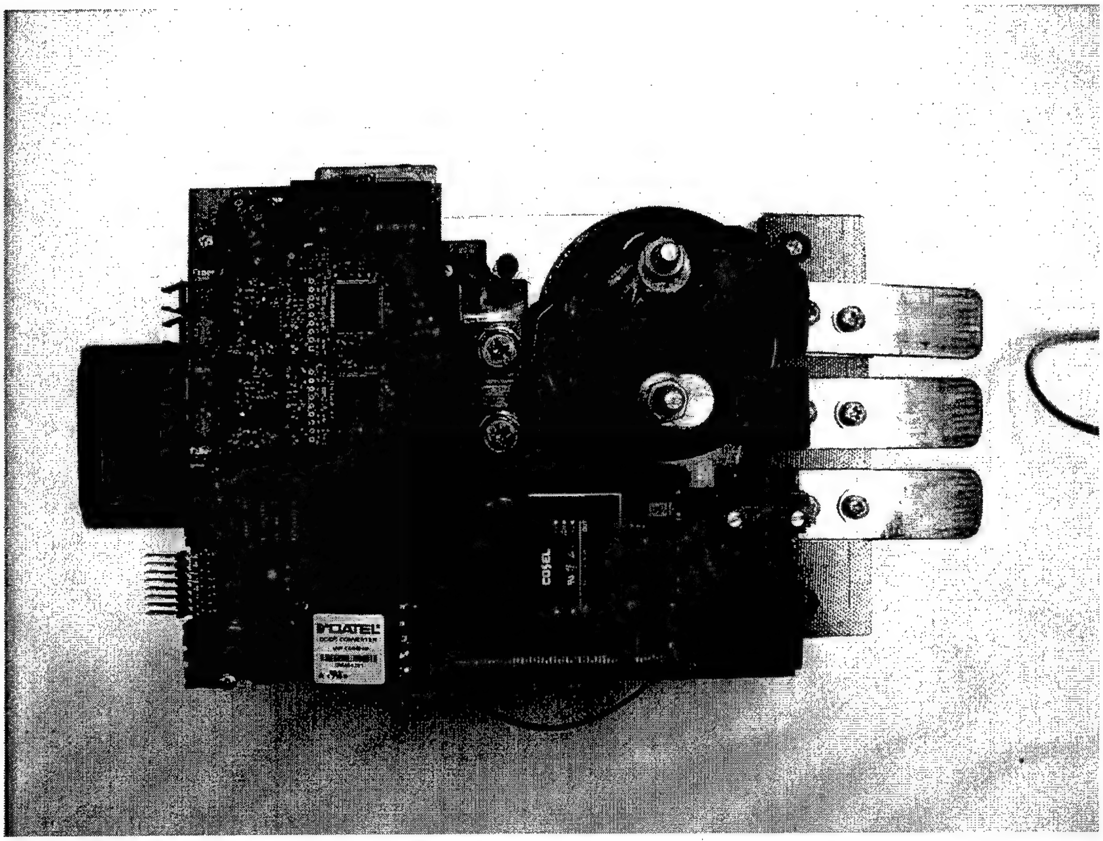


Fig. 4-2 New PEBB module using the Hardware Manager.

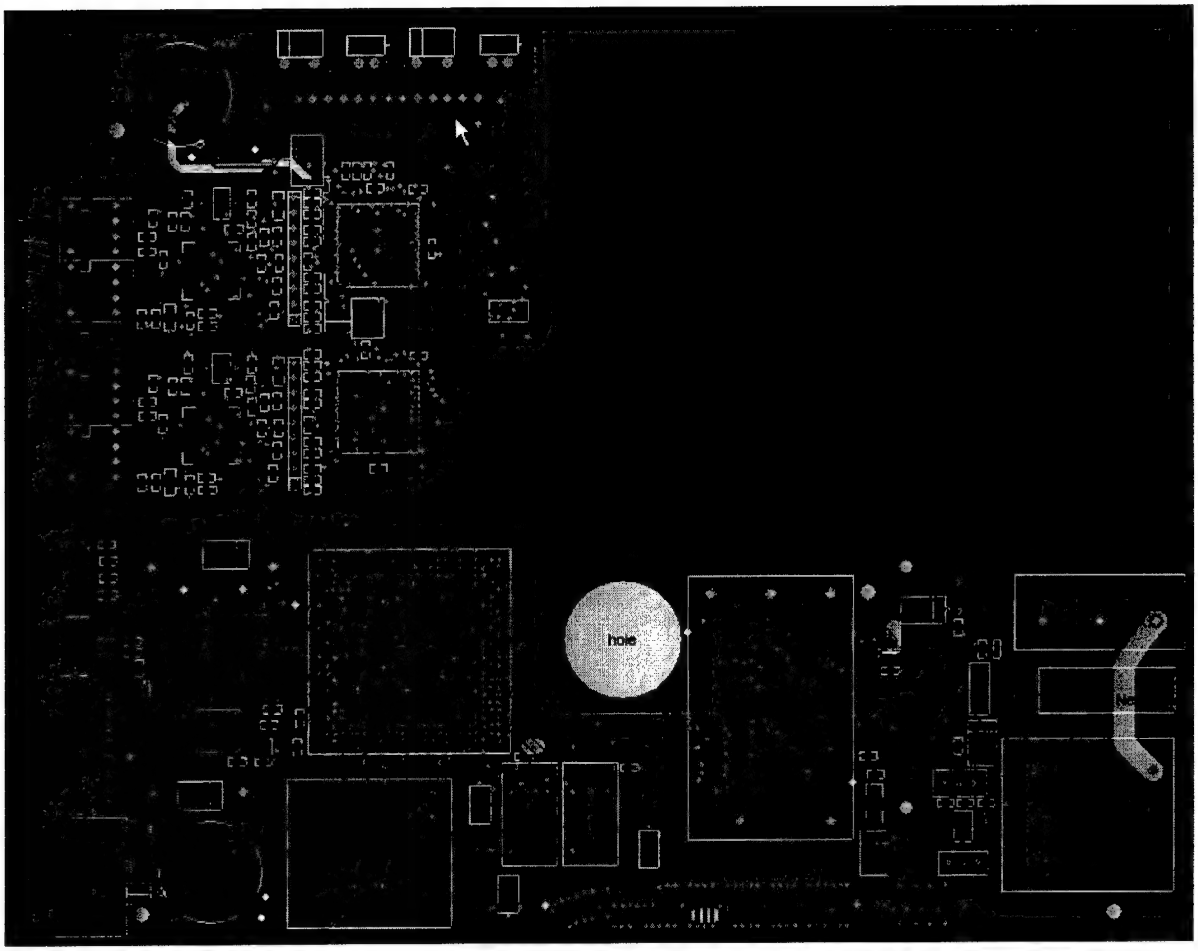


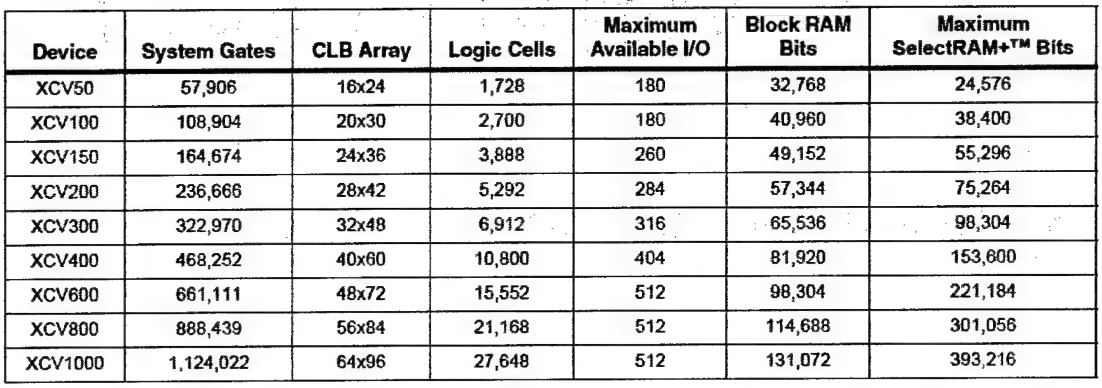
Fig. 4-3 PCB Layout of the Hardware Manager

4.2 Design

The Hardware Manager board interfaces the phase legs to the information system, thus making up the basic power electronics building block (PEBB). It is responsible for performing control functions specific to the hardware used in implementing the power stage, but altogether independent of converter topology. The Hardware Manager, phase legs, and the power stage were designed according to the principles of PEBB, making the system modular and flexible. Different topologies need only control code modifications, and possibly small configuration changes in the power stage.

Several key ideas were kept in mind while designing the new Hardware Manager, such as incorporating sensing and protection mechanisms, the ability to support a dual-ring communication network, and modular design techniques. The Hardware Manager was designed and built to be compatible with the previously built PEBBs and the capabilities of the new generation Universal Controllers. The new Hardware Manager thus includes voltage, current and temperature sensors, dual optical transceiver circuitry, and the IPM driver-interface circuit. The new Hardware Managers are backwards-compatible to the old phase-legs, but mainly designed to take advantage of the capabilities of the new Universal Controller.

Central to the operation of the board is the Xilinx FPGA. This programmable logic chip controls all aspects of the operation of the Hardware Manager, and was chosen because of its ease of use, versatility, and size. The FPGA was programmed with modular VHDL code, which administers every board function, from communication, to sensors, to the PWM signals going to the IPM driver circuit. Because of its size (the number of gates), the FPGA can easily manage all Hardware Manager functions while still operating fast enough to execute all necessary code in one switching cycle. This means the speed of the FPGA was not a determining factor when choosing the switching frequency. As more complex functions and protocols are implemented, it may become a barrier. To solve this, the Hardware Manager may have to use a faster, larger FPGA (please see Fig. 4-4), or use more efficient coding methods.



Device	System Gates	CLB Array	Logic Cells	Maximum Available I/O	Block RAM Bits	Maximum SelectRAM+™ Bits
XCV50	57,906	16x24	1,728	180	32,768	24,576
XCV100	108,904	20x30	2,700	180	40,960	38,400
XCV150	164,674	24x36	3,888	260	49,152	55,296
XCV200	236,666	28x42	5,292	284	57,344	75,264
XCV300	322,970	32x48	6,912	316	65,536	98,304
XCV400	468,252	40x60	10,800	404	81,920	153,600
XCV600	661,111	48x72	15,552	512	98,304	221,184
XCV800	888,439	56x84	21,168	512	114,688	301,056
XCV1000	1,124,022	64x96	27,648	512	131,072	393,216

Fig. 4-4 Virtex-series device comparison

The Hardware Manager participates in the system via dual (or single) optical fiber rings, managed by the PESNet communication protocol. Handling communication are two Cypress communication chips (CY7C9689A-AC) and high-speed optical transceiver circuits. This chip has integrated transmitter and receiver circuits, and is a vast improvement over the communication chips used in the previous Hardware Managers. The Cypress chip is much smaller and thinner, which conserves board real estate, and dissipates heat more efficiently.

Besides being big and bulky, AMD TAXI chips used in the previous version became extremely hot during normal use.

The Hardware Manager has advanced on-board LEM voltage and current sensors, on-board temperature sensor, and a thermocouple input to directly measure the temperature of the IPM. These sensor signals are tied to the FPGA via two analog-to-digital converters.

Powering the board are several compact, low profile dc/dc converters. The board uses a 5V input, and then converts that into 5V, 2.5V, 3.3V, and +/- 15V levels. Furthermore, the board has floating power supplies to drive the gate signals of the IPM (dual 15V, 3000V isolation power supplies).

Last, but not least, the board was outfitted with high-density connectors for the monitoring and debugging of digital I/O signals, as well as to allow for future expansion: an additional sensor board could be connected to the Hardware Manager, and more digital signals sent to the FPGA, expanding its function. The board also has 5 additional A/D channels, allowing more analog signals to be linked to the FPGA.

Overall, the modularity of the PEBB concept has carried over into the design of the Hardware Manager. The structure of the Hardware Manager is shown in Fig. 4-5 , where this can be clearly seen. Since there are no shared data buses, each part is truly independent of the other functions of the board, linked together only at the FPGA.

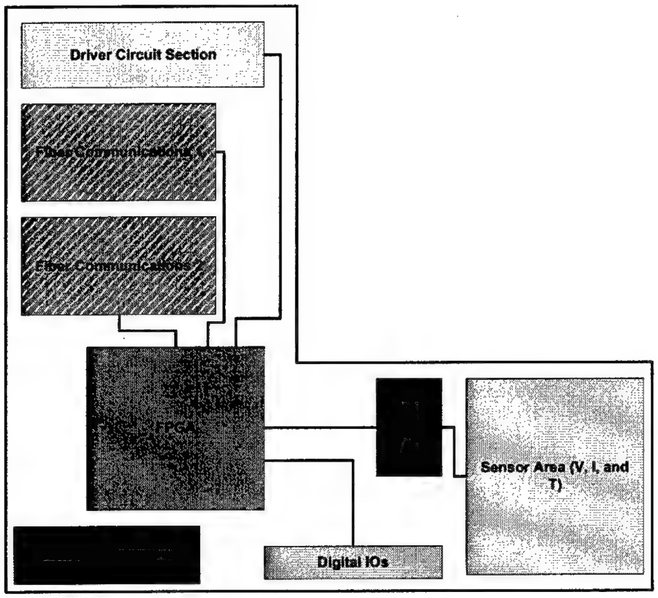


Fig. 4-5 Hardware Manager's modular structure

Section	Functionality
FPGA	Manages board other functions, executes control command.
Communication	Dual-ring compatible, high-speed optical network.
Sensor Area	Voltage, current, and temperature sensors provide information for board monitoring, fault protection, and control code execution.
Digital I/O	Ease troubleshooting by allowing up to 63 digital signals to be analyzed concurrently.
Driver Circuit	Dual, 3000V-isolated floating power supplies. Includes switch fault signal.
Hex Displays	Important debugging and testing tool.

Fig. 4-6 Hardware Manager's sectioned functions

4.2.1 FPGA Design

The FPGA is at the center of the Hardware Manager, both physically and functionally. The Hardware Manager was designed to be modular and sectioned. This modular approach translates into easier troubleshooting, which leads to fast development times. Additionally, this approach contributes to a very organized physical layout and VHDL code. All of the subsections of the Hardware Manager directly interface to the FPGA: the communication chips have direct lines to the FPGA's I/O ports, as do the analog-to-digital converters and the expansion connectors.

The modularity of the VHDL code was very beneficial while troubleshooting. Each section was implemented and tested individually, and the board was verified piece-by-piece. Testing code sections individually narrows down the sources of any encountered problems. Moreover, the debugging connectors can be mapped to virtually any digital signal from the FPGA. These signals can then be seen using a Logic Analyzer.

The specific FPGA chosen was the Xilinx Virtex XCV300-4BG352. This is Virtex series, 2.5V chip, chosen for its compatibility to 5V logic signals. A newer, faster, and more economical alternative would be to use the Virtex-E series; however, because of its incompatibility to 5V signals, nearly all of the input lines would have to be buffered. While simple enough, due to the number of signals between the communication chips and the FPGA, this became expensive in terms of board space. Future revisions of the Hardware Manager may make use of this newer FPGA.

While the FPGA offers a number of benefits, it does so with some difficulties. The manufacturing and populating of the FPGA's BGA package is not a trivial task. The boards have to be etched perfectly, and the vias aligned correctly in order for the component's solder bumps not slide away from the contacts. This was a problem in one specific area of the FPGA where some vias were too close to the pads. However, the solder ball stayed in its place and the signal was preserved. Fig. 4-7, below, shows the detailed FPGA area layout. Problem area is shown in Fig. 4-7. Future versions of the Hardware Manager will explore better manufacturing technology, which will allow the use of smaller vias. This single adjustment should help enormously with the mounting of the FPGA.

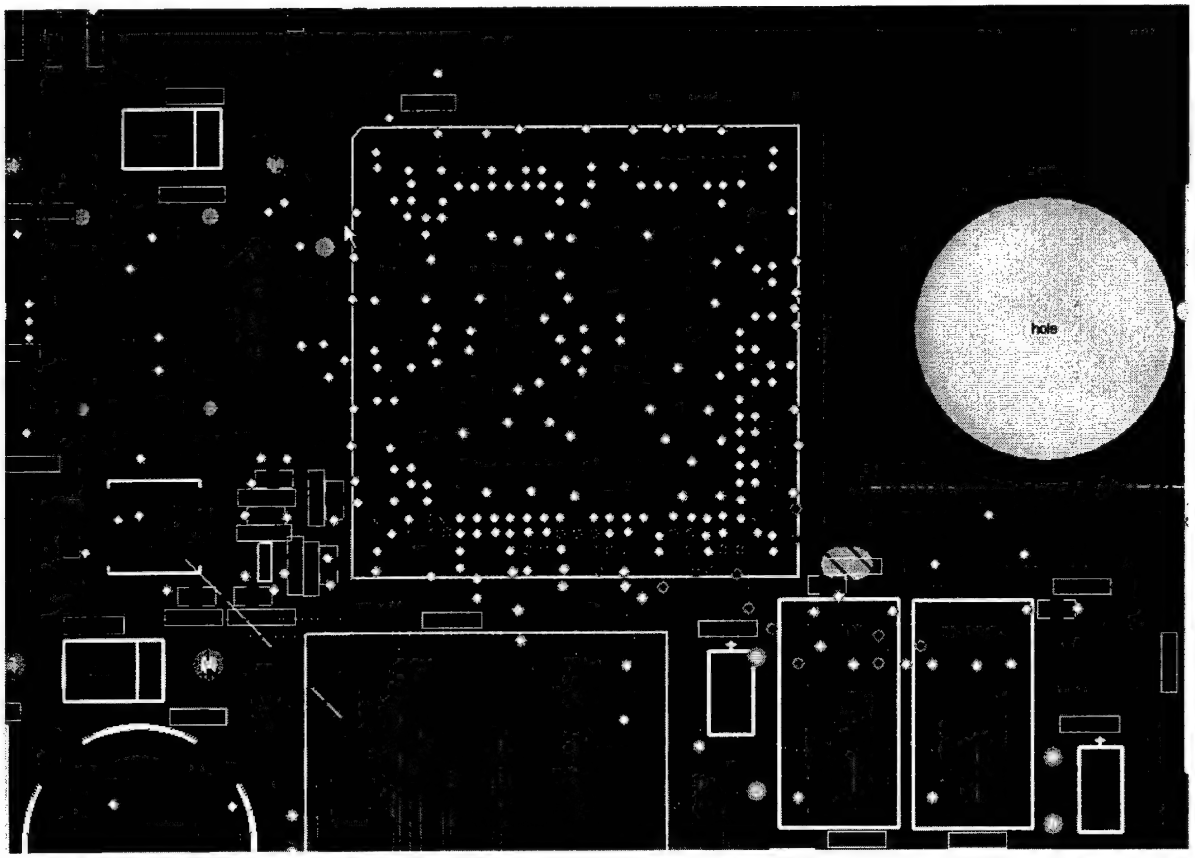


Fig. 4-7 Layout of the FPGA

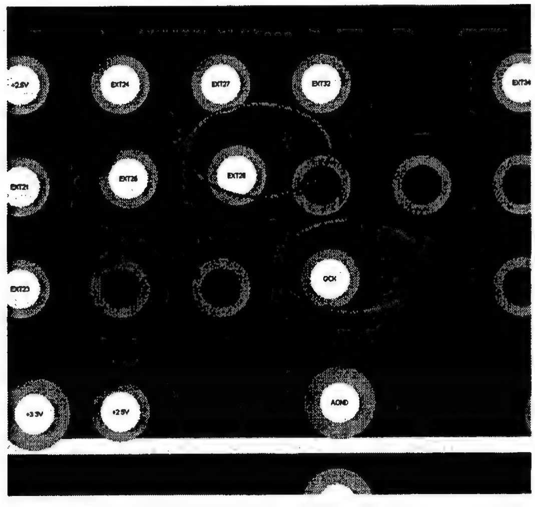


Fig. 4-8 Areas of concern in mounting of the FPGA

4.2.2 Communications Circuitry Design

The communications circuit design was a crucial portion of the Hardware Manager. The concept of the PEBB relies on the ability of the Hardware Managers to communicate with the Universal Controller and with other Hardware Managers. Another reason the communication circuitry needs to be robust is possible interference from high-power switching signals from the IPM, which is located not too far from the digital circuitry of the Hardware Manager. Clearly, without communication, the Hardware Manager would be rendered useless.

The Cypress communication chip was chosen for several reasons: integrated transmitter and receiver, small TQFP package, and good thermal characteristics. The Cypress chip was a vast improvement over the AMD TAXI chips used in the previous Hardware Managers, and offered many more advanced options. Another reason for choosing the Cypress chip was its compatibility to the old AMD chips via a simple adapter board (see below, TAXI2CYP). However, the Cypress chips also caused some headaches, mostly due to the complexity of its functions.

The communications circuitry used in the Hardware Manager and Universal Controller are high-speed optical networks, which require well-built transceiver circuits. Trace thickness and length become very important at these high frequencies, as well as placement of components. Therefore, great care was used in designing this portion of the board, and the lessons learned from the latest version of the Universal Controller were very useful. Still, a few problems were encountered, especially relating to the Cypress transceiver chips. Several pins in the Cypress chip take on different active states depending on the mode of operation. Failure to recognize this led to several errors, which delayed launching of both the Universal Controller and the Hardware Manager.

More problems were found with the termination circuit used to bias the differential signals: IN+, IN-, OUT+, and OUT-. The magnitude and biasing of these signals are keys to the well being of the communication between modules. Line termination resistors incorrectly biasing the waveform caused initial failures with the receiver. The erroneous waveforms had smaller than normal amplitudes, and as consequence, data was not recognized by the transceiver chip. After the termination was corrected, and the bias adjusted, communication was established. The correct termination biasing circuit is shown below, in Fig. 4-9 , for both receiver and transmitter:

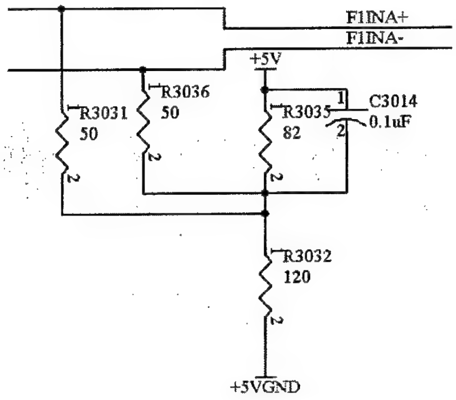


Fig. 4-9 Termination Circuit for Differential Signals in both Receiver and Transmitter Circuits

The complete receiver and transmitter circuit is shown in Fig. 4-10 and Fig. 4-11 , respectively. The layout of the communication section of the Hardware Manager is detailed in Fig. 4-12 .

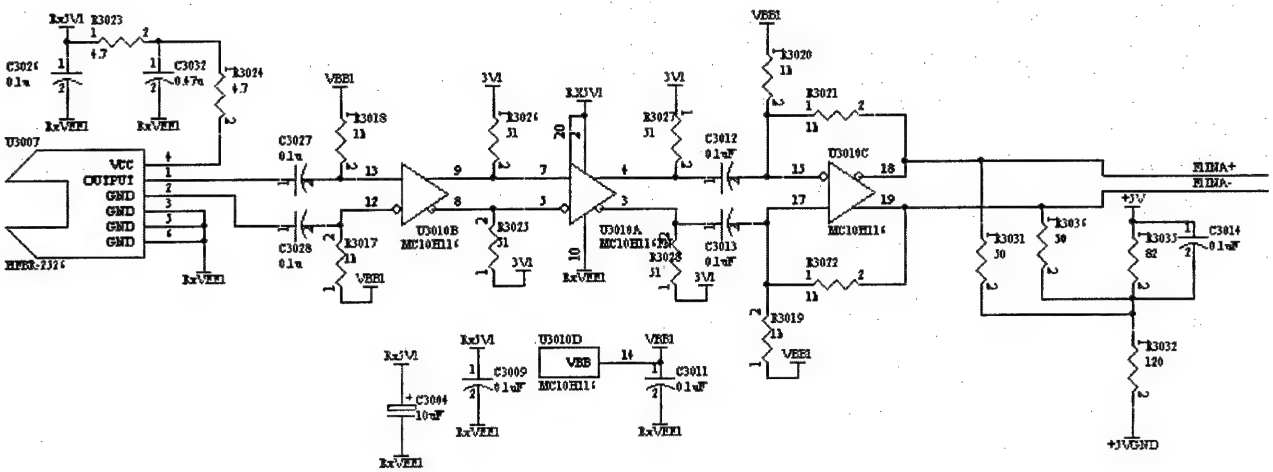


Fig. 4-10 Optical Receiver Circuit

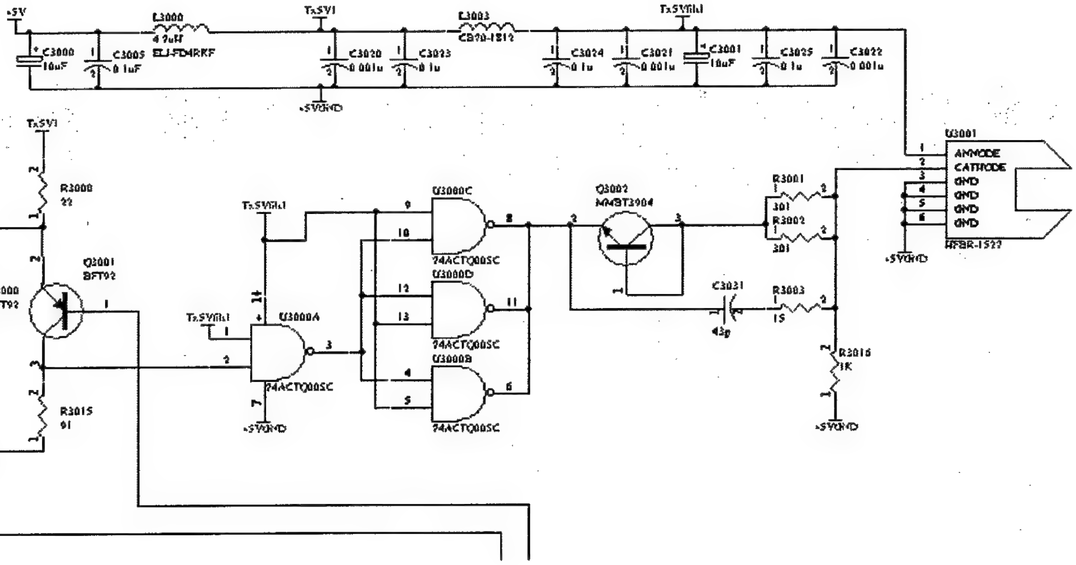


Fig. 4-11 Optical Transmitter Circuit

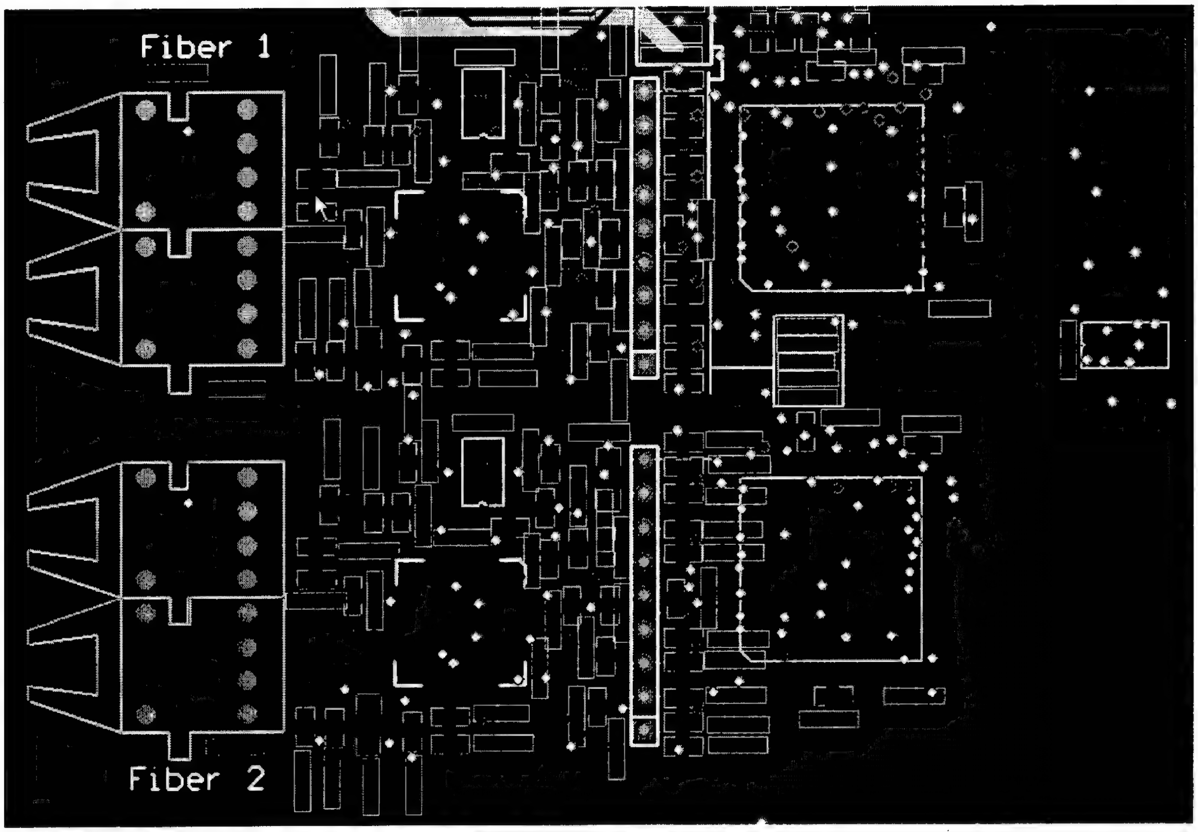


Fig. 4-12 Layout of the communications circuitry

As mentioned above, the Cypress chip had an additional advantage in its ability to be easily interfaced to the old, and discontinued, AMD TAXIchip AM7968 and AM7969 (receiver and transmitter). Therefore, an interface board was created that effectively translated the Cypress chip to a TAXIchip interface. The layout of this board, called TAXI2CYP, is shown below, in Fig. 4-13 .

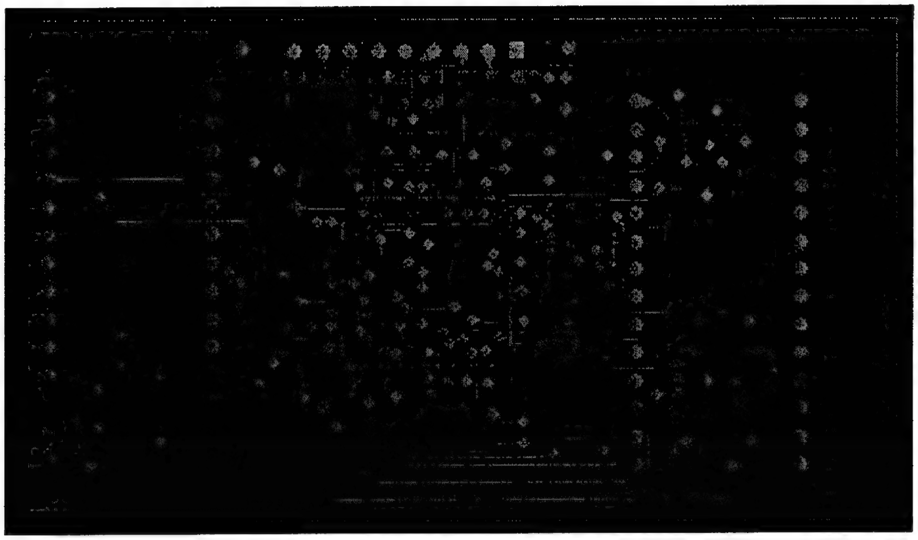


Fig. 4-13 Composite Print of TAXI2CYP Board

4.2.3 Sensor Circuitry Design

The Hardware Manager was designed with on-board voltage, current, and temperature sensors. The voltage and current sensors used are LEM LV-25P and LA-200, respectively. The voltage and current sensor information is sent to an Analog Devices AD7869 analog-to-digital converter, which in turn sends the 12-bit data to the FPGA.

The voltage and current sensors are accurate, top-of-the-line parts. Both sensors are closed-loop Hall Effect sensors, with roughly 100-kHz bandwidth. Having accurate and reliable sensors is very important, since the control algorithm acts on the values of voltage and current supplied by these sensors. The voltage and current sensors are connected as shown in the schematic diagrams below (Fig. 4-14 and Fig. 4-15).

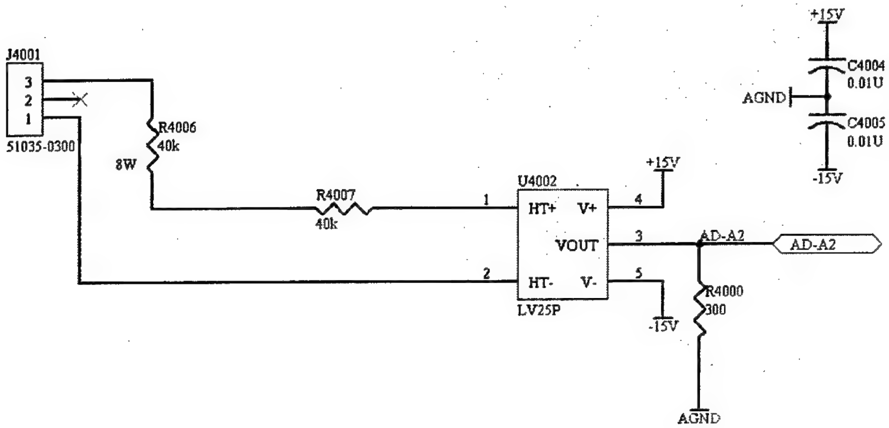


Fig. 4-14 Voltage sensor schematic diagram

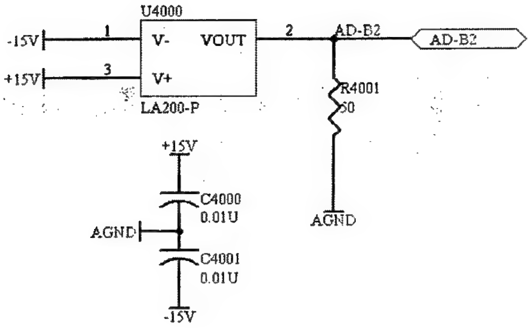


Fig. 4-15 Current sensor schematic diagram

The temperature sensor is a Maxim MAX6627 SPI-compatible device. This device uses the temperature dependence of the resistance of a silicon device, such as an MMBT-3904 transistor, to accurately measure on-board temperature (see schematic on Fig. 4-16). The purpose of this sensor is to monitor the sensing circuitry. As shown above, the voltage sensor uses two 5W, 40-kΩ resistors placed in series to measure the dc-bus voltage. At full rated power, these resistors each will dissipate close to 4W of power. Should these resistors overheat, they may cause damage to sensitive digital components. Therefore the monitoring of the on-board temperature becomes an important safety mechanism, allowing the Hardware Manager to prevent damages due to over-temperature.

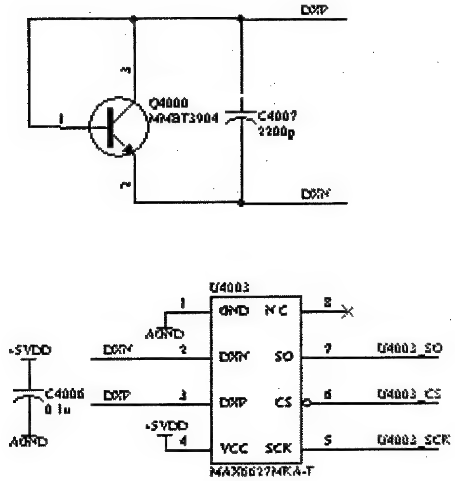


Fig. 4-16 Schematic diagram for the Temperature Sensor

4.3 Operation

The Hardware Manager was designed with the intent to operate in plug-and-play fashion, whereas any PEBB module can take place of a faulted one, or operate in any region of a specific topology. The Hardware Manager was built to be application independent, and can be used in multiple topologies. The only restriction is their individual power rating of 33kW, 800V$_{dc}$, and 50A nominal current. The Hardware Manager also offers protection and warning systems to go along with system-level redundant safety measures.

Operation of the Hardware Manager is controlled by the Xilinx FPGA. The FPGA may be programmed directly (via JTAG), or by the on-board PROM. The normal operation of the Hardware Manager includes managing communication with other nodes, sensing temperature, voltage, and current levels, and executing the corresponding control instruction sent by the Universal Controller for that switching period. The Hardware Manager receives a specific command from the Universal Controller, executes it, and reports back its sensor values – it knows nothing about the present topology, control algorithm, or power level, which are all handled in the Universal Controller.

In our system, each hardware manager controls one half-bridge IPM module (Fig. 4-17). The IPM module contains two switches (top and bottom) and their respective gate drivers. The interface to the module is logical with a single ended power supply of 15 volts (see below). The IPM module connects directly to the PCB and is isolated from the digital circuitry via opto-couplers. The gate drive scheme provides short circuit protection (by means of the de-saturation voltage circuit) and fault detection, with the fault signal 'f'. In order to reduce the number of supply voltage levels (and thus the number of dc-dc converters), V_{dd} and V_{cc} were arranged in floating point configuration.

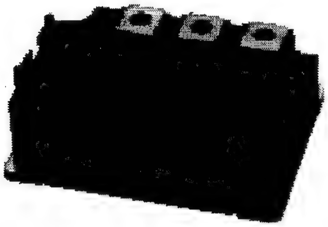


Fig. 4-17 IPM Module

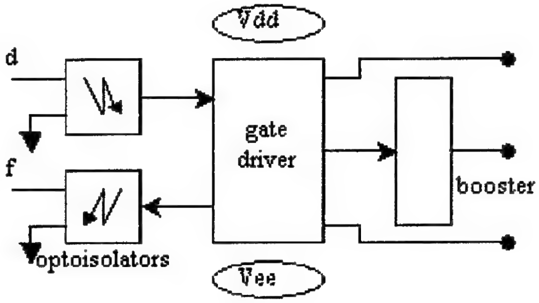


Fig. 4-18 Dual Module IGBT Gate Drive Circuit

The normal operation of the Hardware Manager includes the execution of the control algorithm sent by the Universal Controller: the controller sends a duty cycle, which is modulated, and applied to the switches above. The information sent over the ring is tailored for that individual Hardware Manager. The Hardware Manager, in turn, sends its sensor information back to the Universal Controller, which computes the next duty cycle value. This is a simplified version of the normal communication over the fiber ring; other information may be sent, as necessary. The details can be found under the new PESNet communications protocol, PESNet 2.0.

The Hardware Manager boards have several fault protection and warning systems. The VHDL code includes over-voltage, temperature, and over-current protection, as well as status LEDs, which let the user know it is operating properly. These LEDs show the on-board dc/dc converters are supplying the appropriate voltage level, as well as FPGA and IPM faults.

A worst-case power consumption analysis of the Hardware Manager yielded a value of about 15W. However, under normal usage, the board only requires about half of that figure. The worst-case power consumption figure is so large mainly because of the FPGA, which does not use all of its pins at one time. Based on this analysis, the input current to the Hardware Manager is limited to 3A by a resettable dc fuse.

4.4 Testing

Testing of the new Hardware Manager has been very positive in all aspects, with no major setbacks. Fig. 4-19 shows a picture of the new PEBB while running experimental tests. Errors included easily correctable, non-threatening cosmetic mistakes, such as small footprint inaccuracies. Board level tests have verified all components of the Hardware Manager. The waveforms shown in Chapter 5 show the PWM signals sent to the switches. They represent the midpoint of the IPM, and the control signals to the bottom and top switches. This small test verifies the operation of the driver circuit and PWM generator.

The Hardware Manager was also tested in a single phase, dc/dc converter configuration, connected in a loop with a Universal Controller (please see Fig. 4-20 and Fig. 4-21). The resulting waveforms are shown in Chapter 6. This test enabled us to test the power stage (to low power levels), and all functions of the Hardware Manager, including communication. The duty cycle was sent from the Universal Controller via fiber. It was possible to change the duty cycle on the fly, and watch the Hardware Manager's immediate reaction. The Universal Controller received sensor information back from the Hardware Manager.

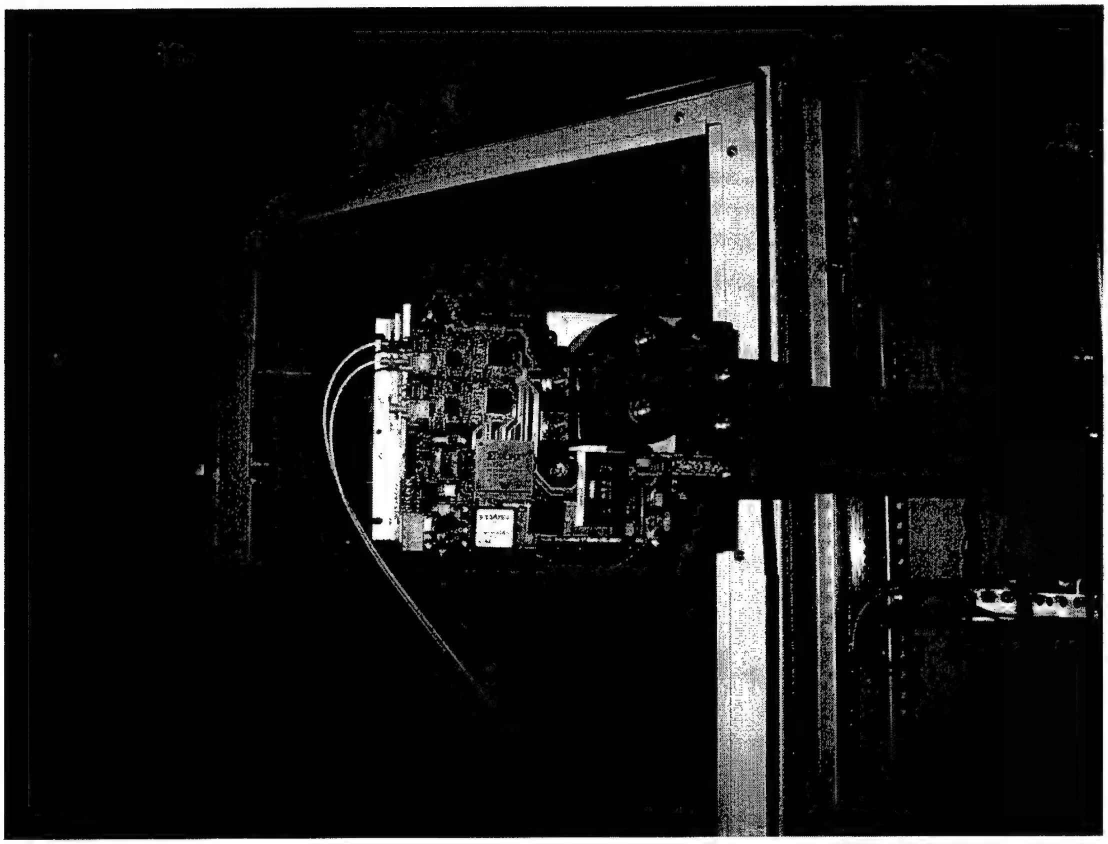


Fig. 4-19 New PEBB module mounted on the removable slide featuring the Hardware Manager and main power connectors.

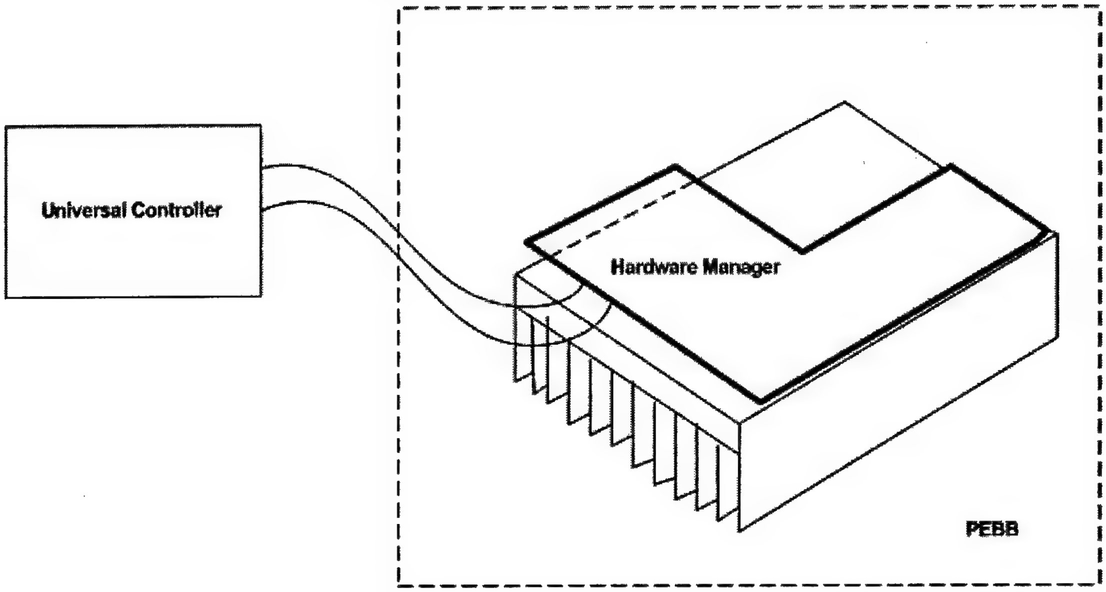


Fig. 4-20 Universal Controller in loop with one PEBB

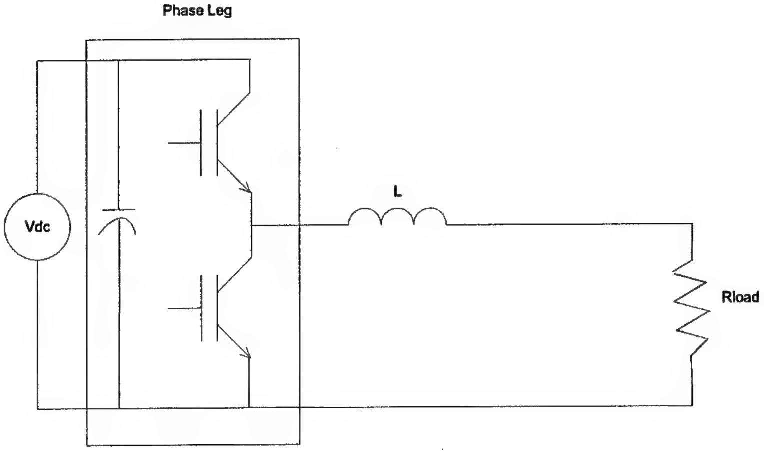


Fig. 4-21 Single phase dc/dc converter configuration

Although this was a simple test, conducted at a fraction of the power capabilities of the PEBB, the results were very encouraging. This test showed communication can be established on the fiber link, and that the Universal Controller is able to send and receive data from the Hardware Manager. A further test was executed with two phase legs (one new and one old) in a loop, in the following configuration:

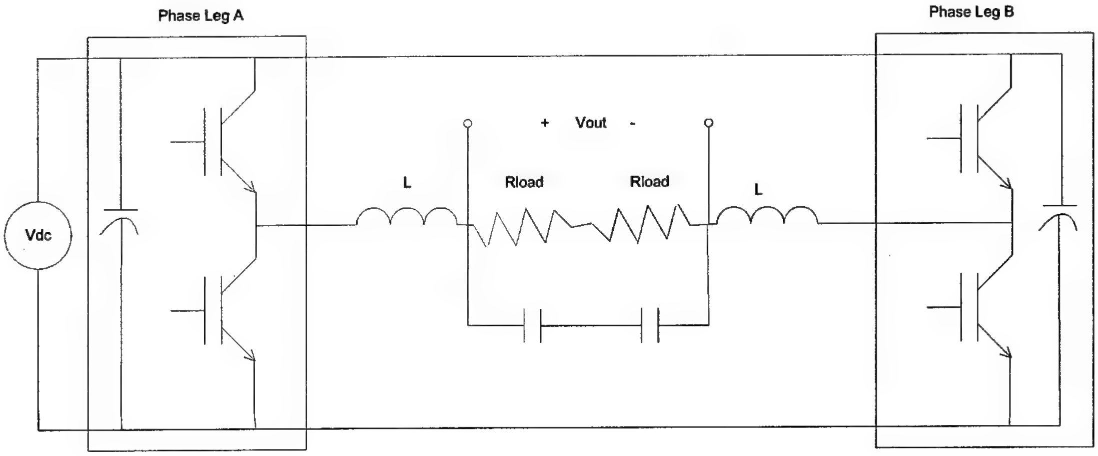


Fig. 4-22 Full bridge inverter configuration

4.5 Future Research

The diagram in Fig. 4-23 shows the structure of the PEBB system, and all of its interfaces. Future research involving the Hardware Manager will concentrate on its interface with the switch (i_y). For current power levels, less than 100kW, the interface between the Hardware Manager and the phase leg has been defined and studied in this project. However, for higher power applications where single devices may be larger than the board itself, the Hardware Manager would not efficiently interface the PEBB. The structure of the current PEBB may need to be changed in order to accommodate higher power applications – so that it can be ever more flexible. Additionally, studies will concentrate on making the Hardware Manager independent on the switch used. If this interface is characterized and standardized from both sides, the device used will no longer be directly related to the Hardware Manager, and vice-versa, as long as both are compatible with the interface.

Changes that would make our PEBB more flexible include moving the high-frequency dc link capacitor and sensors away from the Hardware Manager. They would be fitted instead into their own separate intelligent modules (smart passives and smart sensors). These and other alterations would open way to a highly-desirable, multi-megawatt PEBB.

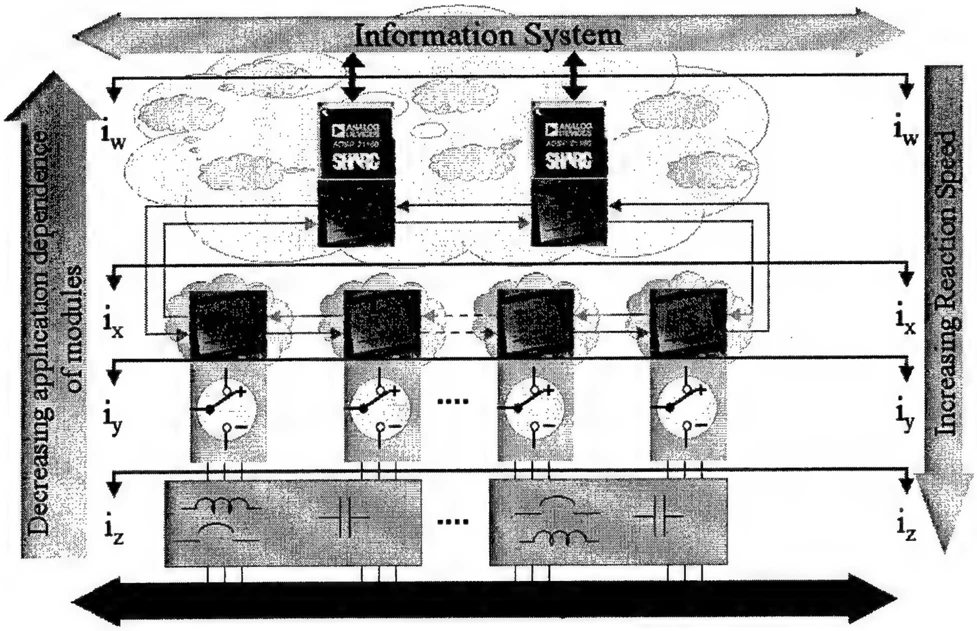


Fig. 4-23. Interfaces of the Power Electronics Building Block (PEBB)

4.6 Conclusion

The concept of the Power Electronics Building Block rests in the ideas of reliability, flexibility, and modularity. The PEBB allows design in power electronics to be done in a standardized manner, instead of custom design jobs. This leads to faster development time, easier troubleshooting, and lower costs. The Hardware Manager has been designed with all of these ideas in mind, in the way to becoming a true PEBB. It has achieved reliability through the use of multiple on-board sensors and protection mechanisms (on top of system level protection), topology and power level independence, and partitioned design, both physically and in software. Future research on the Hardware Manager and the PEBB concept will strive to understand all of its boundaries, and thus expanding the PEBB definition, and how it applies to the Hardware Manager.

5 PEBB-BASED POWER STAGE

5.1 PEBB Partitioning Studies

Unlike modern digital technology, which utilizes an array of developed components or cells to build a system, modern power systems lack a high degree of integration and standardization. As a result, designers are often forced to build entire systems from scratch each time, which is costly in engineering time as well as system reliability. In order to remedy this situation, the concept of PEBB has been developed. These building blocks are integrated power modules serving a function, which is commonly found in a wide number of power systems. Depending on the instructions given to the controller, the PEBB can function as, for instance, a DC/DC converter, an AC inverter, a synchronous rectifier, or a motor controller. In fact, it can do any of these jobs interchangeably, depending only on the instructions given to it. The goal of the PEBB development is to create a power-processing component that moves most of the design away from specific circuit topology consideration and power electronic switch and associated inductors, capacitors, and other ancillary component selection, up to a systems level. As PEBB modules can be connected together to form several power system topologies, greatly reduced design efforts as well as increased system simplicity and reliability are achieved. In addition, maintenance cost is reduced since individual modules are easily replaced and the number of stock spare parts is reduced. PEBBs can be also the best choice to minimize both the layout and packaging parasitics, because all the power semiconductor devices, control circuits, and the busbar will be integrated together as a large power device.

To evaluate a fully functional PEBB two main fields must be investigated: identifying the PEBB topology for construction of the more standard converter topologies and the applicability of distributed control to the multi-cell topology, which includes the design of the communication link and controller hardware required for cell control. In particular, one aspect still need investigation is how to design, distribute and partition into PEBBs the hardware components usually present in customized power converters.

5.2 Architecture of PEBB-Based Power Electronics Systems

A power electronics system is a set of power processing devices governed by a control system that uses and produces the information about the operation of the power stage. Therefore, that information-power characterization can be represented as a composition in a bi-dimensional space. With the power switch in the origin of coordinates the two dimensions extend horizontally and vertically as schematically shown in Fig. 5-1. With the information in the vertical dimension, the different levels of control authority build upon the controlled power units. The power is represented in the horizontal direction. Along that direction, the power components (semiconductors, passives and auxiliaries) exchange power between them and the power system. Previous works analyzed the PEBB in its information dimension and based on that proposed a distributed control architecture [xv].

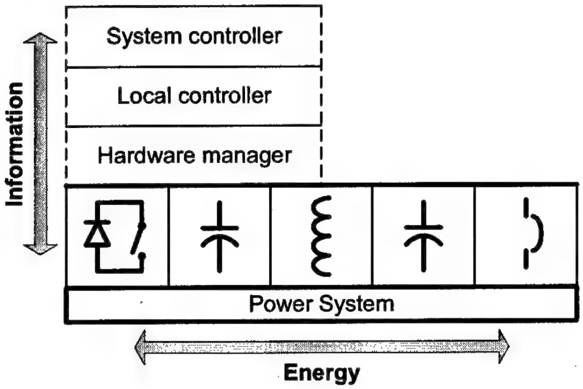


Fig. 5-1 General PEBB Architecture

The PEBB is build upon two composing structures: one is related to the power management and is composed by the switching devices and the related passive and auxiliary components; the other is the control that manages the proper operation of the power equipment. The control structure basically process information related to the operation of the system while the power components actually process the energy. Therefore these two "dimensions" of the PEBB: energy and information can be arranged in a two-dimensional space as shown in Fig. 5-1. The vertical is the information dimension and can be analyzed like an OSA information network producing a layer characterization of the PEBB. In such layer classification, the power equipment belongs to the bottom layer with the different control levels build upon it. On the other side, the energy dimension characterizes that bottom layer and is horizontally represented in Fig. 5-1 along this dimension, the power equipment: semiconductors, passives and auxiliaries exchange power between them and the power system. The previous figure is only a conceptual representation of the PEBB in its basic characteristics because it exist quite a difference between the ways the power electronic systems handle the information and the energy. For example there is a significantly high interaction between the power system components and a layer partition does not exist along the energy dimension. Nevertheless, there are also some similarities that will be shown in this work.

5.2.1 Functional analysis and characteristics of the information flow in PEBB

The functional characterization of the PEBB control architecture has already been proposed, Fig. 5-2 shows the PEBB figure of merit along the information dimension.

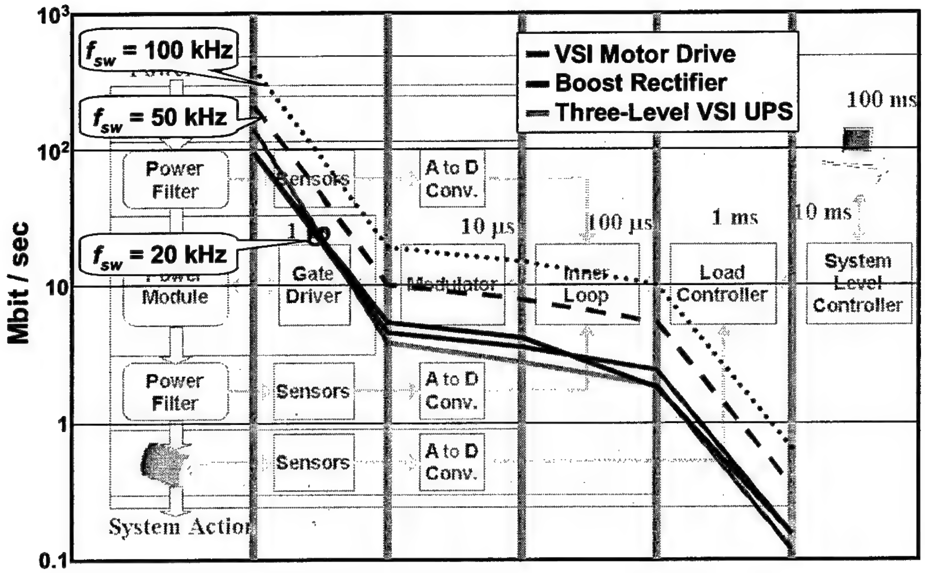


Fig. 5-2 Information capacity at the different levels of control

Although the layers and interfaces are not clearly defined in the energy dimension, the system characteristics in such dimension can be analyzed following a similar path as the one used for the information. That analysis included a quantification of the information transmission capacity (Mbit/sec). In the energetic characterization the quantification is related with the amount of energy that the different components handle. This magnitude is not the power on the devices, nor their losses because there is no interest in an efficiency evaluation here. In general, the characterizing magnitude is the product of a voltage and a current. It can be a reactive power for passive components, but in some cases, as the power semiconductors, is the product of two magnitudes that are not present at the same time.

We can call the product of the V and I magnitudes used for the evaluation, the power capacity, or power merit. The proposed evaluation method is better clarified using as example the calculation of the power capacity at the different components in the PEBB. Power semiconductor devices: its capacity is evaluated by the required V inverse capacity, which is Vll, and the rms value of the conducting current, Irms

P = Vll x Irms

DC capacitor: it operates at the dc voltage level, Vdc, and is circulated in the permanent regime by the ripple current. Therefore its power merit is:

P = Vdc x Icrms

AC inductor: it is circulated by the converter line current, and the current ripple creates a voltage ripple at the inductor terminals.

P = Vlrms x Irms

AC capacitor: it operates at the line-line rms voltage, and although there can be a relatively large AC current, our interest is in the ripple current created by the operation at the switching frequency. Therefore

P = Vll x Icrms

Auxiliary components: including protection devices, start or stop auxiliary circuits, etc. They are not operative during at the permanent regime; so the average power is comparatively low. On the other side, the power requirement during the operation is important. Therefore, it is necessary to distinguish between the average and instantaneous requirements, but both are relevant for the evaluation. Fig. 5-3shows the evaluation of the power capacity of the components of a phase leg that has the following nominal characteristics (taken form PnP project):

rated power	P=33kW
Dc link voltage	Vdc=800V
switching frequency	fsw=20kHz
Ac side main frequency	fac=60Hz
line to line rms voltage level	Vll=480V
Ac rms current	Il=120.3A
Ac peak current	Ilpk=170.1A

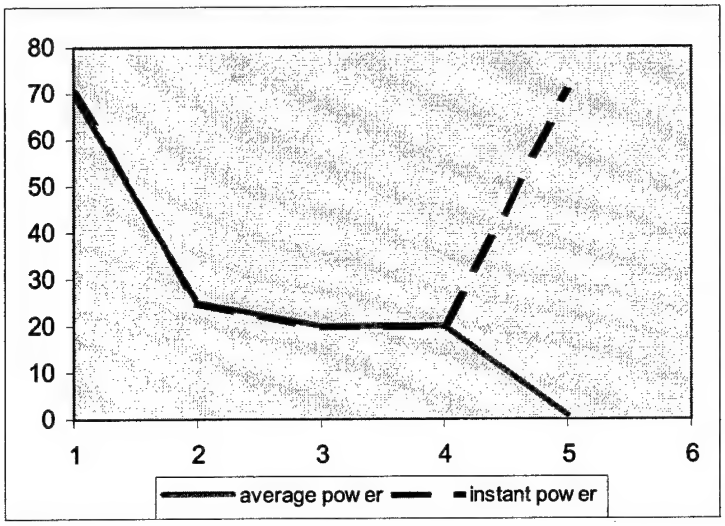


Fig. 5-3 The energy figure of merit (similar to something?, ha) The abscissa refers to: 1 semiconductor devices, 2 DC capacitor, 3 line inductor, 4 line capacitor, 5 other auxiliary components

5.2.2 Physical boundaries of the PEBB

The useful concept of PEBB imposes some challenges on its technological implementation enabling a widespread use in practice. One of those challenges is on finding the physical boundaries of the PEBB. The calculation procedure of a PEBB involves the evaluation of the electric requirements of the system components for all the possible applications. This evaluation is necessary for dimensioning and selecting the semiconductor, passive, and auxiliary devices. In that procedure, it is observed that while some applications impose a large requirement on a component, another one may not even require the use of such a component. This shows the fact that there exists some power capacity in the different components of the PEBB that is not fully used through the different applications. We can evaluate such non-usable power capacity at a particular component as the difference between the maximum and minimum capacity requirement. Exemplifying, the semiconductor devices are usually required to be fully utilized in all the applications while the AC inductors are not required for some of them. Moreover, we can refer the mentioned unusable capacity to the maximum requirement, usually the value adopted for construction, and express it as a percentage. This is shown in Fig. 5-4 (in magenta) as well as the combination (in blue), by multiplication, of the power merit (in yellow) and the percentage of unusable power. That blue line represents the amount of power capacity that is non-usable at the different components of the PEBB.

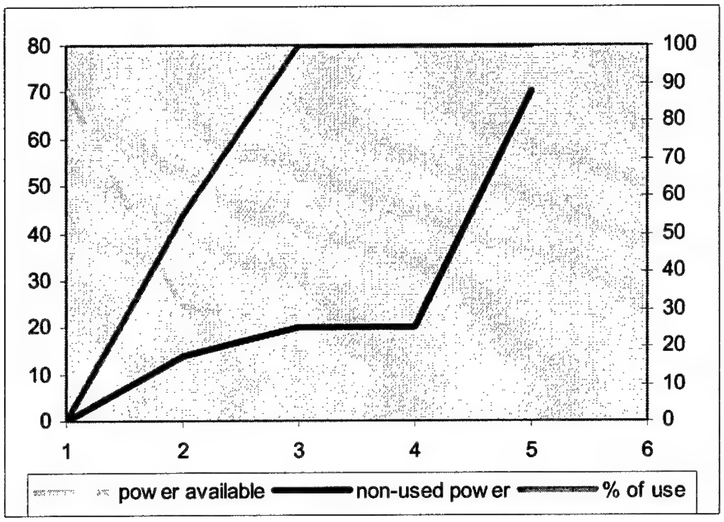


Fig. 5-4 Calculation of the unusable power. Yellow: power merit at the different components; magenta: percentage of non-usable capacity; blue: product of the previous two magnitudes giving the non-usable power at the system components

The concept of the unusable power capacity can be employed to set (or at least to discuss about) where the boundaries of the PEBB in the energy dimension must be. Cost considerations prevent from having a lot of non-usable power on any power system. For the PEBB construction this means that if a component non-usable capacity is large, it is probably worth to not include it in the PEBB and add it only for those applications where it is required. Therefore, there is a limit for such unemployed power capacity. Components with an unusable power capacity below such limit are probably to be included in the PEBB and the ones above the limit are to be considered as auxiliary equipment added in case the application requires them. The described idea is represented in Fig. 5-5. The limit of unusable power capacity is arbitrary and chosen based on technical-economical considerations. The proposed idea is consistent with existing practices of power electronics module construction. For small power applications, the affordable un-usable capacity is comparatively large (the relative position of the limit line in the chart is high) and the PEBB includes a lot of power components. For large power applications, the limit of unusable capacity (placed relatively low in the chart) will leave out of the PEBB almost all devices, and in such cases the PEBB includes only the power semiconductors.

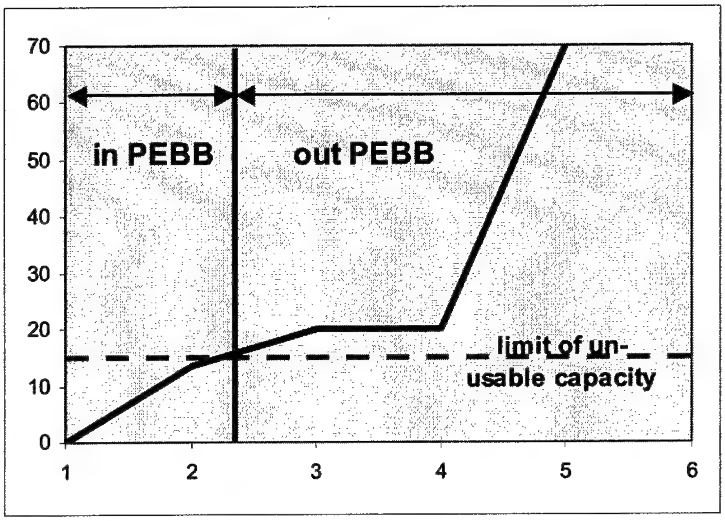


Fig. 5-5 Non-usable power characteristic and PEBB boundary based on the limit of affordable non-usable capacity

5.2.3 Control Characteristics of Power Electronics Systems

The information dimension in a power electronics converter is characterized mainly by the functional and temporal requirements of the control system. The functional requirements are all the control functions that the system needs to do in order to achieve its desired operation given its configuration. On the other hand, the temporal requirements mainly address the relation amount of information-time, measured in Mbit/sec, at the different parts of the control system given the operation requirements like switching frequency and modulation.

The analysis of different control functions required in a power electronics system shows a set of common functions at the low control authority level and an increased differentiation when growing in authority level. This is general and valid in the wide range of power conversion systems. As an illustrative example a functional analysis of typical applications for utility systems is shown in Table 5-1[xvi][xvii],[xviii].

On the other side, the evaluation of the information flow along the different control stages shows a large capacity requirement close to the power conversion that becomes smaller when moving away from the energy conversion components. The characteristics from this functional-temporal point of view make the power electronic systems appropriate for implementation of distributed control architecture. In [xix], such kind of architecture is proposed; it also has three levels of authority: hardware manager, local controller and system controller. In a more general sense, a power electronics system has different layers with different functions and characteristics. A system built by combination of building blocks constitutes then a network of such blocks that operates under a distributed control system with layering characteristics. Therefore, systems built on PEBB can be matched with the OSI network standard; this new vision enhances the previously proposed architecture.

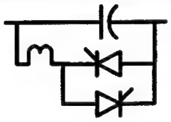
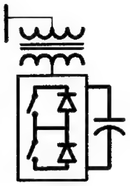
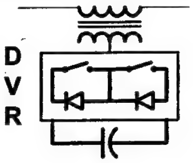
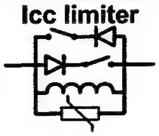
5.2.4 PEBB Stage Characteristics

From the hardware point of view, the PEBB concept establishes that its composition includes a set of components common to that PEBB range of applications, which is intended to be as wide as possible. Therefore, in addition to the power-devices, the building block incorporates passives, auxiliary components, and a set of low-level control functions common to many applications.

A review of medium and high power converter topologies shows two families of widely used phase leg structures. One is the phase leg composed by two active switches with its respective anti-parallel diode (scheme A in Fig. 5-6). It can be used in AC/DC, DC/AC, and DC/DC converters, such as boost rectifier, two level and multi-level voltage source inverter (VSI), and full/half bridge converter. The other commonly used phase leg consists of two active switches with diodes in series with each of them (scheme B in Fig. 5-6); it can be configured as buck rectifier or current source inverter. Other phase leg variants are shown in schemes C and D in Fig. 5-6. The proper operation of the discussed topologies requires the presence of voltage or current source characteristics at the different switch terminals. Therefore, capacitors or inductors must be connected at the proper terminals to complete the basic switching cell. Auxiliary elements like overvoltage protections or connection switches may also be included.

The PEBB is completed with the inclusion of the components necessary to do a set of low-level functions that include signal power amplification, level shift, isolation, protection and diagnostic functions. These components are the gate drivers, transducers, A/D converters, and optical and communication interfaces. A local controller (hardware manager) is then attached to the power-switching cell to accomplish and supervise these functions. In the system being developed these functions are handled by means of a FPGA that also handles the communication to and from higher-level control and other PEBBs. In this sense, the PEBB functions as a computer-controlled power-switching unit or a power processor.

Table 5-1 Functional Analysis of some power electronics applications in utility systems

	5.2.4.1.1 Application	Control	Conversion
TCSC 	- Load flow control - Transient Stability enhancement - Power oscillation damping	- Synchronization - Voltage (or Z) control - Firing angle computation	- Switching control (modulation control) - Pulse gating - Safe commutation enabling (dv/dt, di/dt limit) - Primary device protection - Power magnitudes sensing and conditioning
STATCOM 	- Voltage support - Reactive compensation - Transient stability enhancement - Power oscillation damping	- Synchronization - Current Control - Vdc control - Duty cycle computation	
DVR 	- Sag mitigation - Voltage regulation - Energy storage administration - Connection by-pass	- Synchronization - Voltage control - Current control - Vdc control - Duty cycle computation	
Icc limiter 	- Short circuit current limitation - Connection / disconnection	- Synchronization - I, dI/dt control - Firing angle computation	

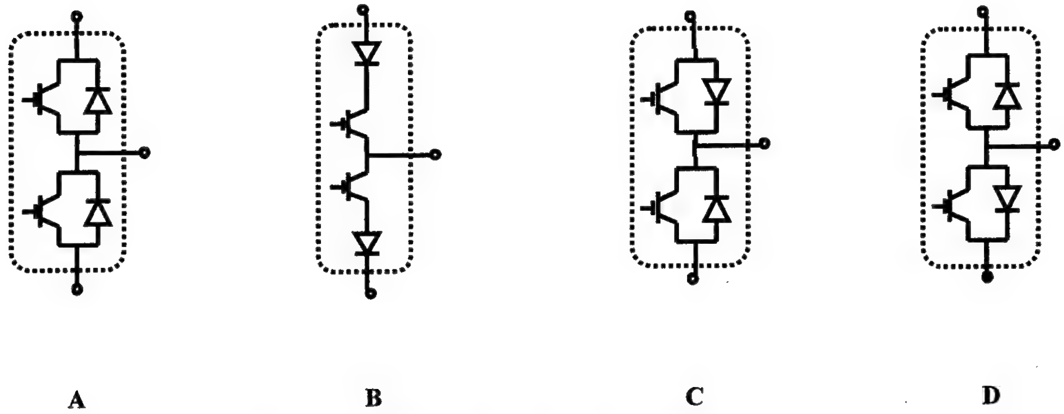


Fig. 5-6 PEBB Phase Leg Structures

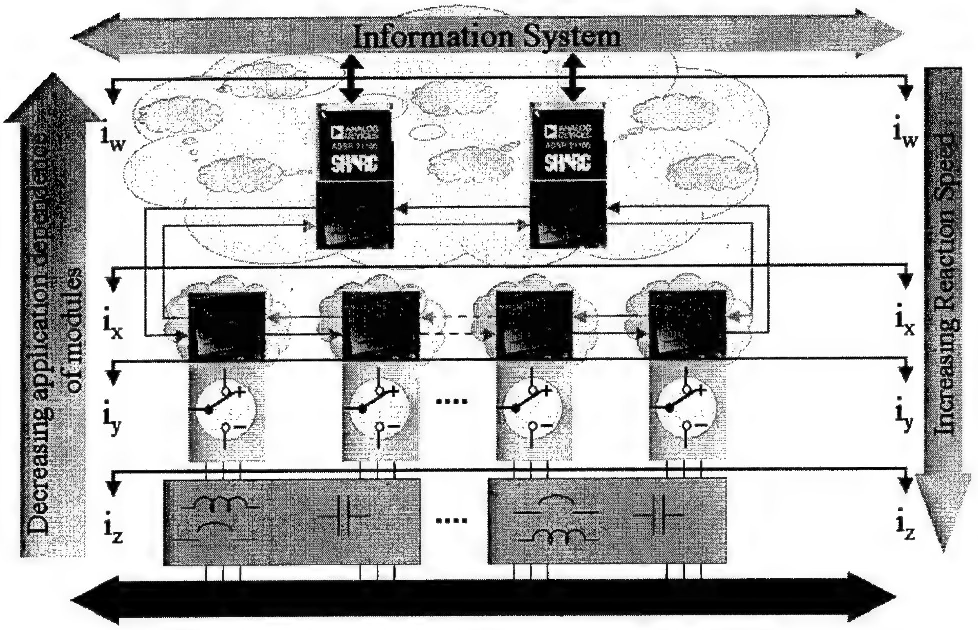

Information System
Decreasing application dependency of modules
Increasing Reaction Speed
i_w
i_w
SHARC
SHARC
i_x
i_x
i_y
i_y
i_z
i_z
....
....

Fig. 5-7 Interfaces of the Power Electronics Building Block (PEBB)

5.3 PEBB-Based Power Stage Developed

The power stage of the integrated hardware/software PEBB system envisioned by the PnP concept is shown in Fig. 5-7. The main goals sought by adding plug and play capacity to the PEBB modules are to allow for easy reconfiguration providing great modularity, and improved accessibility for service and maintenance. Among the desired features is hot swappable control and communications for true PnP control software. This goal basically considers the spatial distribution of the PEBB modules, as their placement within a constraint space will allow or not to attain the easy reconfiguration. This basically calls for a mechanical design effort, which we approached in the following way.

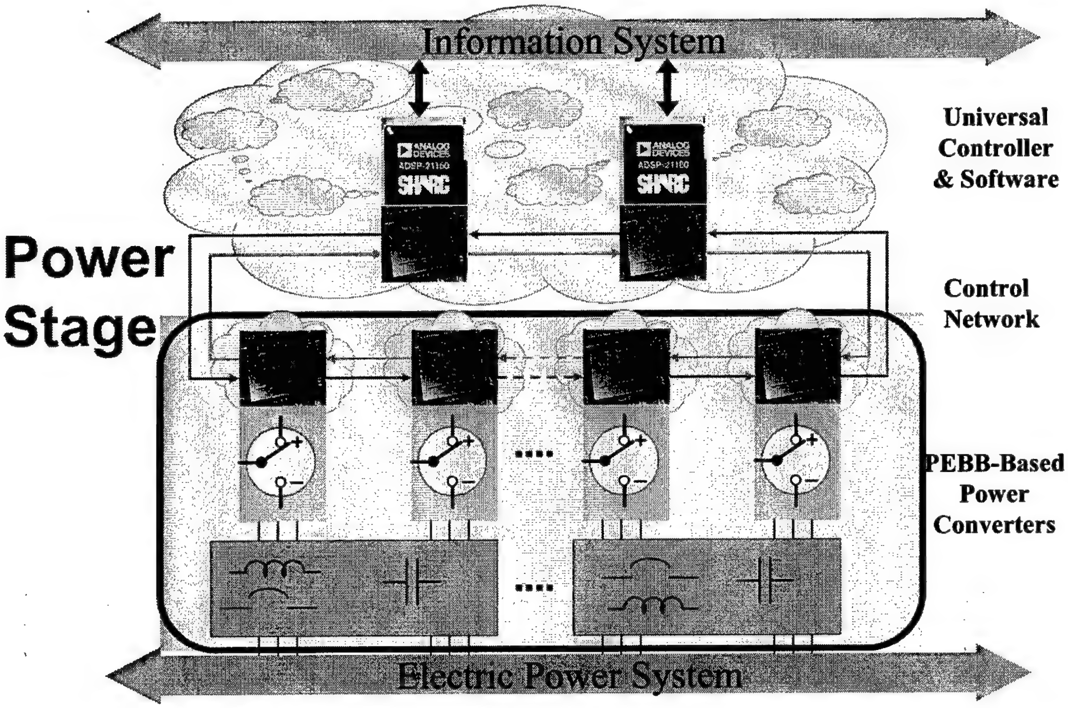

Information System
Universal Controller & Software
Power Stage
Control Network
PEBB-Based Power Converters
Electric Power System
ANALOG DEVICES ADSP-21160 SHARC
ANALOG DEVICES ADSP-21160 SHARC

Fig. 5-8 Power stage subsystem of the Plug and Play demonstration system.

Fig. 5-8 shows the PnP System structure proposal, featuring a functional, temporal and spatial distribution partitioning. The power stage itself encompasses all 3 dimensional distributions, the spatial being perhaps the most complex given the direct interaction of all power components, PEBB, busbars, contactors, protection devices, etc. In a conventional converter these elements are fixed, but for the PnP approach, with distinctive separate elements it becomes significantly more intricate to come up with a clean, modular solution. Think only in the complexities associated to reconfigure the converter module as shown in the different topologies depicted in Fig. 5-9 . In these topologies particularly we defined PEBBs as the common element within the voltage source topologies under study. From observing the vast majority of high-power topologies these element turned out to be a phase-leg, two IGBT connected in series forming a single-pole double-throw switch. This basic building element solely considers the converter switching action, with whom however all existent topologies may be built directly using the proposed PnP System

Architecture, the PNC 3-level is the only one that requires further development as its basic switching element is now comprised of two PEBBs, which must operate under strict synchronization to operate properly (PESNet).

The PEBB concept as such provides the means to simplify the power processor, switching element interconnection, however the converter structure is composed as well by bus bars, filtering elements, contactors and protection devices, dc-link bus with caps or batteries in UPS systems, not to mention communication buses, low power distribution for control systems, and cooling, forced air or any other depending on the power rating. Consequently, the converter still remains a fairly structured system, and actual reconfiguration still demands more than simply reordering some modular blocks.

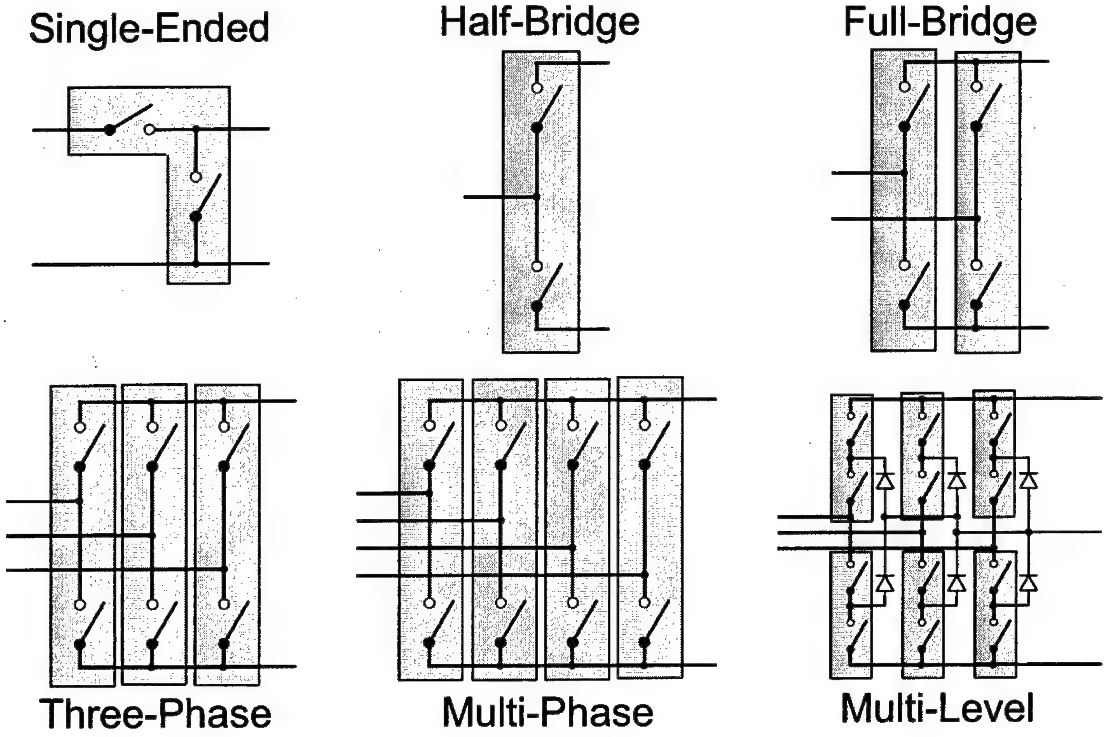


Fig. 5-9 PEBB-based topologies for the Plug and Play power system.

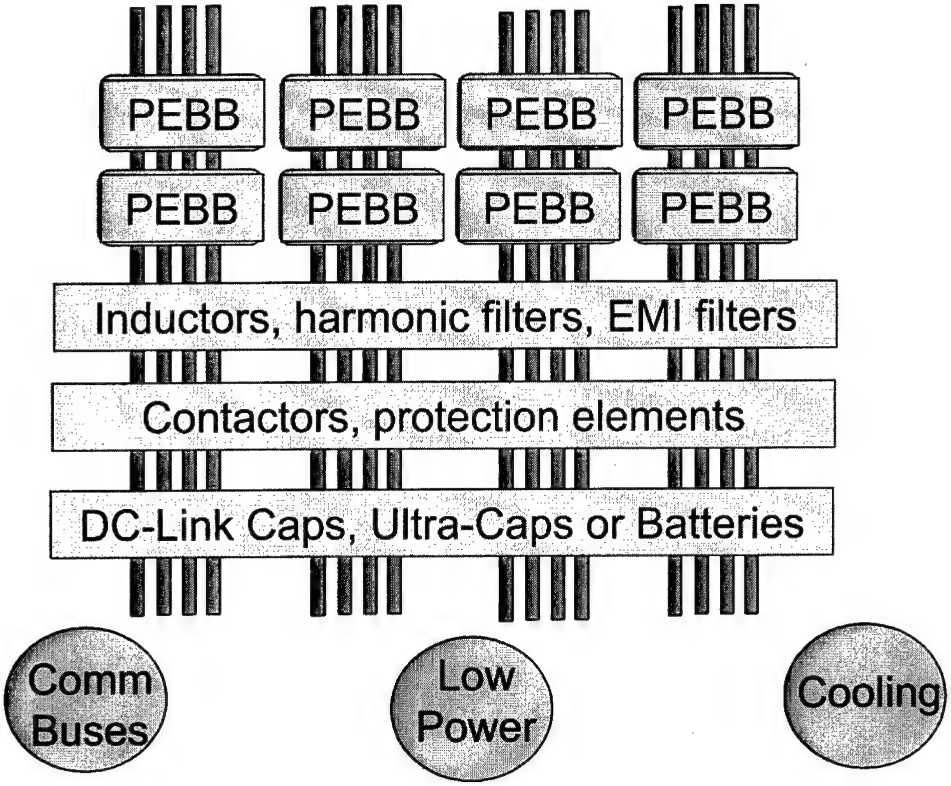

PEBB
PEBB
PEBB
PEBB
PEBB
PEBB
PEBB
PEBB
Inductors, harmonic filters, EMI filters
Contactors, protection elements
DC-Link Caps, Ultra-Caps or Batteries
Comm Buses
Low Power
Cooling

Fig. 5-10 Power system schematic partitioning of the different components pertaining to the Plug and Play physical structure system.

With this in mind we redesigned the converter cabinet structure so that it would have such an organization that would allow taking the PEBB concept a step further, by identifying passive PEBB modules within, as could be ac inductors with integrated protection, sensors, contactors, as well as capacitors with the same features. This is shown in Fig. 5-10. What follows is a more detailed presentation of the design of each of these components.

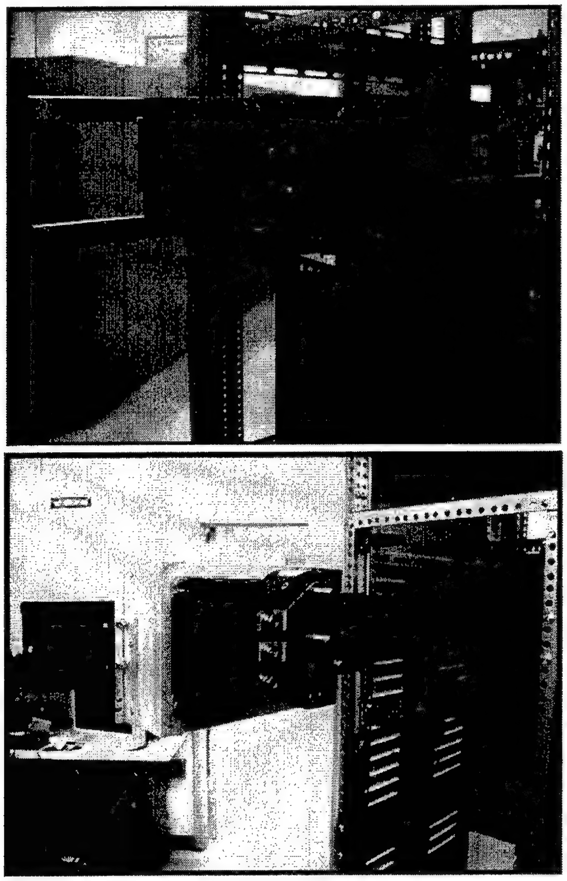


Fig. 5-11 The original fixed-ball bearing slides used to mount the phase legs were later replaced by removable single slides for ease of operation.

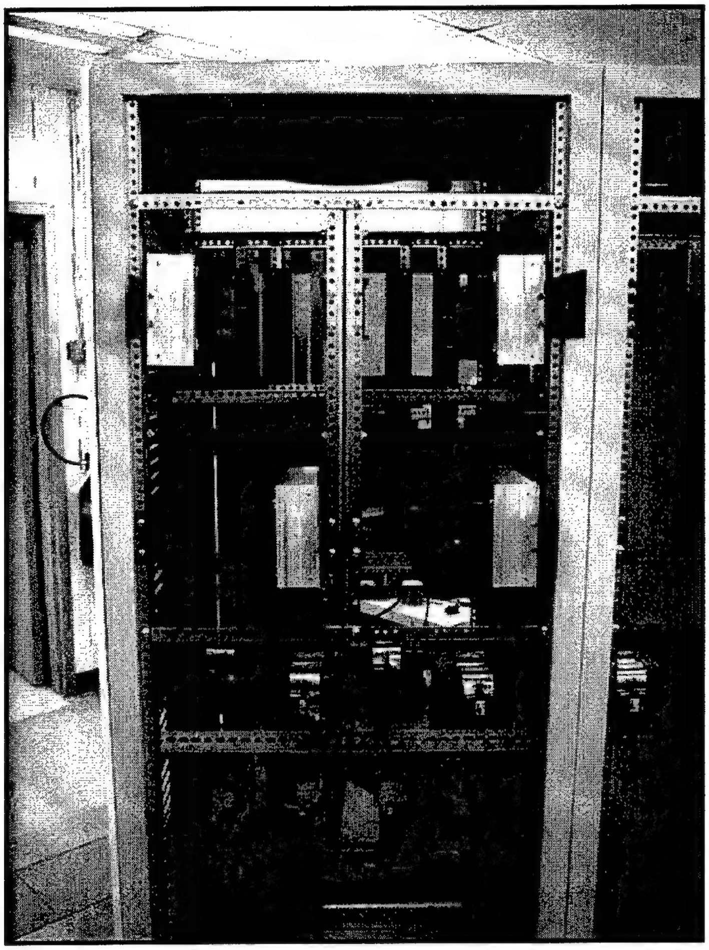


Fig. 5-12 Frontal view of cabinet.

5.3.1 Semiconductor Device Selection and Heat Sink Design

The electric characteristics of the applications for the phase-leg under investigation are as follows:

rated power P=100kVA;

dc link voltage Vdc=800V;

switching frequency fsw=20kHz;

ac side main frequency fac=60Hz.

Assuming Vll = 480V as line to line rms voltage level, the rms and peak values for the ac current are:

ac rms current Il=120.3A;

ac peak current Ilpk=170.1A.

When a maximum peak-to-peak ripple of 25% with respect to ac peak current is considered, the maximum current flowing through the power switch becomes:

ac max current Ilm=191.36A.

On the basis of the reported values, the selection of the power devices for the phase leg should be carried out in the 1200V – 300A IGBT range. Three different options are here considered for the IGBT module selection: dual module IGBT, dual intelligent power module (IPM) IGBT, six-pack intelligent power module (IPM) IGBT. The third option is to be accomplished by arranging in parallel connection both the top switches and the bottom switches, this option has been considered because the six-pack IPM IGBT is more commonly product than the dual IPM IGBT. However, parallel connection of IGBTs is critical concerning about static and dynamic current balance among the paralleled devices; then further recommended current deratings of 15-20% should be applied.

The dual module IGBT usually requires 2 optoisolators, 1 gate driver, 1 booster configuration, and 2 isolated power supply voltage levels in order to drive each switch (Fig. 5-13). The gate driver can be either the hybrid gate driver suggested and produced by the same company of the selected IGBT or a IGBT universal driver (MC33135). However, the MC33135 and similar components, desirable in order to design a more general purpose drive board, accept a maximum voltage level equals to 20V thus allowing a negative voltage supply Vee=-5V (the positive voltage supply must be set at Vdd=15V); whereas the manufacturing companies suggest to provide Vee=-10V (–5V≤Vee≤-15V) in order to minimize turn-off switching energy. The gate drive scheme accomplishes the short circuit protection (by means of the de-saturation voltage circuit) and fault detection, providing the fault signal f. In order to reduce the number of supply voltage levels (and then the number of dc-dc converters), Vdd and Vee could be arranged in floating point configuration.

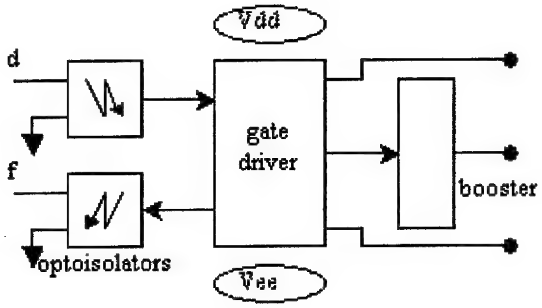


Fig. 5-13 Dual Module IGBT Gate Drive Circuit

The gate drive circuit for each switch of an IPM configuration is shown in Fig. 5-14. In this case neither external gate driver nor booster circuit are needed, as well only 1 power supply voltage level (+15V) is required. IPMs have built-in protection circuits that prevent the power devices from being damaged should the system malfunction or be overstressed.

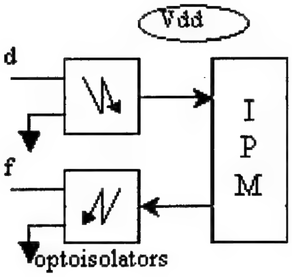


Fig. 5-14 IPM IGBT Gate Drive Circuit

Fault detection and shut down schemes that allow maximum utilization of power device capability without compromising reliability have been developed. Control supply under-voltage, over-temperature, over-current, and short-circuit protection are all provided by the IPM's internal gate control circuits. A fault output signal f is provided to alert the system controller if any of the protection circuits are activated. The short-circuit protection uses actual current measurement to detect dangerous conditions. This type of protection is faster and more reliable than conventional de-saturation protection schemes.

On the basis of the previous descriptions and comments on IGBT modules and gate drive configurations it seems to be reasonable the choice of the dual IPM IGBT for the application under investigation. A research on power semiconductor devices throughout the web-sites of the major companies put on evidence that only Powerex-Mitsubishi and Toshiba produce dual IPMs in the 1200V – 300A range. Thus, the components PM300DVA120 (Powerex-Mitsubishi) and MIG300Q101H (Toshiba) are considered in the following in order to design the cooling system.

5.3.2 Power Losses

The total power losses in the power semiconductors of a phase-leg are composed of conduction and switching losses. Conduction losses are the losses due to the device's conduction characteristics. The conduction losses are a function of the currents in each particular device and that device's dc electrical characteristics. The switching losses are a function of the switching frequency, the current in each device, and the device's dynamic characteristics.

The design of the phase-leg under investigation should be accomplished in order to allow the building of several converter topologies (at the same power rating) and the adoption of different modulation techniques. The switch and diode current, and thus the power losses, depend a great deal on the modulation techniques used for the selected application [4]. For sake of generality, continuous modulation (CM) techniques are preferred for the phase-leg power losses determination as discontinuous modulation (DCM) techniques (as dc-bus clamping) provide a reduction of the switching power losses (even if higher distortion concentrated in the tops of the waves partially denies this advantage), whereas the conduction losses are almost the same of CM techniques.

In the present report, data-sheets based methods for calculations of power losses are considered; extensions of these methods can be used in case of multi-level power converter configurations. When experimental testing and measuring system are available, extended models based on extraction of parameters can be used.

5.3.2.1 Conduction Losses

In order to simplify the calculation of switch and diode currents, the ac phase current can be assumed to be sinusoidal and the switching frequency is assumed sufficiently higher than the ac fundamental frequency. Equations for the average and RMS switch and diode currents are derived based on a quantitative analysis of the current waveforms. Conduction losses in the semiconductor devices are then approximated using a piece-wise linear approximation of the device's on-voltage characteristics. The typical IGBT voltage/current graph Vce/Ic is approximated by the following linear equation:

$$V_{CE} = R_{on-s} \cdot I_C + V_{CE0}$$

$$R_{on-s} = \frac{V_{CEn} - V_{CE0}}{I_{Cn}}$$

where VCEn and ICn are the rated (manufacturers catalogue) current and the collector-to-emitter voltage at the rated current, whereas VCE0 is the threshold voltage. At the same way, the diode forward voltage characteristic can be approximated by means of a linear law with the origin at the threshold voltage VF0:

$$V_F = R_{on-d} \cdot I_C + V_{F0}$$

$$R_{on-s} = \frac{V_{CEn} - V_{CE0}}{I_{Cn}}$$

where VFn is the diode voltage drop at rated current.

The following equations define the conduction losses for phase-leg switches and diodes:

$$P_{on-s} = I_{av-s} \cdot V_{CE0} + R_{on-s} \cdot I_{RMS-s}^2$$

$$P_{on-d} = I_{av-d} \cdot V_{F0} + R_{on-d} \cdot I_{RMS-d}^2$$

where Iav, IRMS and Ron are the average current, the RMS current and the conduction resistance of respectively the switch (s) and the diode (d).

The phase current for 3-phase applications usually lags the phase voltage by the phase angle φ; because the current is assumed a simple sine function, the math works out to be easier if we define the voltage leading the current by φ and integrate over the current waveform. The ac phase current, the duty cycles of the phase-leg top switch and bottom diode are defined according to the following equations (under constant frequency conditions):

$$i_l = I_{lpk} \cdot sin\theta$$

$$d_s = \frac{1}{2} \cdot [1 + Msin(\theta + \varphi)]$$

$$d_d = 1 - d_s = \frac{1}{2} \cdot [1 - Msin(\theta + \varphi)]$$

where M is the modulation depth (represent the normalized voltage and is between 0 and 1) and θ is the phase angle. At full modulation the duty cycle varies from 0 to 100%.

Assuming the switch current is fairly constant over one modulation cycle, the average current over that cycle is the current times the duty cycle. The average switch current is calculated over one half of the full sine wave and can then be found by integrating the current times the duty cycle. The reason for averaging the switch current over one half of the period is found in the estimating the device junction temperature rise. In fact, for applications in which devices are completely off (and then they don't produce power losses) for half a period, each device can be approximated to be subjected to a long train of equal amplitude (half a period) load pulses; then, the transient thermal impedance should be considered in order to calculate the device's over-temperature over each half a period. Thus, the function ac phase current times the duty cycle is integrated from 0 to π and then divided by π:

$$I_{av-s} = \frac{1}{\pi}\int_0^{\pi} i_l \cdot d_s \cdot d\theta = \frac{1}{2\pi} I_{lpk} \int_0^{\pi} \{\sin\theta[1 + M\sin(\theta+\varphi)]\} \cdot d\theta$$

thus resulting

$$I_{av-s} = I_{lpk}\left(\frac{1}{\pi} + \frac{M\cos\varphi}{4}\right).$$

Similarly, for the diode it is found:

$$I_{av-d} = I_{lpk}\left(\frac{1}{\pi} - \frac{M\cos\varphi}{4}\right).$$

The RMS current is found by first squaring the current, then integrating and taking square root of the resultant. The RMS value of a pulse waveform is found by squaring the amplitude and integrating over the on time, therefore the duty cycle is not squared. The switch RMS current is

$$I_{RMS-s} = \sqrt{\frac{1}{\pi}\int_0^{\pi} i_l^2 \cdot d_s \cdot d\theta} = I_{lpk}\sqrt{\frac{1}{2\pi}\int_0^{\pi}\left\{\sin^2\theta[1 + M\sin(\theta+\varphi)]\right\} \cdot d\theta}$$

and then

$$I_{RMS-s} = I_{lpk}\sqrt{\frac{1}{4} + \frac{2 \cdot M\cos\varphi}{3\pi}}.$$

In the same way, it is found for the diode:

$$I_{RMS-d} = I_{lpk}\sqrt{\frac{1}{4} - \frac{2 \cdot M\cos\varphi}{3\pi}}$$

The average and RMS values for the bottom switch and top diode are the same. The equations for average and RMS currents are a function of φ; setting φ equal to 0 and M equal to 1 will maximize the switch current and minimize the diode current.

If sine modulation with the 3rd harmonic is considered, then it is found:

$$I_{av-s} = I_{lpk}\left(\frac{1}{\pi} + \frac{\sqrt{3} \cdot M\cos\varphi}{6}\right)$$

$$I_{av-d} = I_{lpk}\left(\frac{1}{\pi} - \frac{\sqrt{3} \cdot M\cos\varphi}{6}\right)$$

$$I_{RMS-s} = I_{lpk}\sqrt{\frac{1}{4} + 2 \cdot M\frac{30\cos\varphi - \cos 3\varphi}{45\sqrt{3} \cdot \pi}}$$

$$I_{RMS-d} = I_{lpk}\sqrt{\frac{1}{4} - 2 \cdot M\frac{30\cos\varphi - \cos 3\varphi}{45\sqrt{3} \cdot \pi}}$$

5.3.2.2 Switching Losses

Switching loss is the power dissipated during the turn-on and turn-off switching transitions. In high frequency modulation, switching losses can be substantial and must be considered in thermal design. The most accurate method of determining switching losses is to plot the IGBT current and voltage waveforms during the transitions. Multiply the waveforms point by point to get an instantaneous power waveform. The area under the power waveform is the switching energy expressed in Joule/pulse. There are pulses of power loss at turn-on and turn-off of the IGBT. The instantaneous junction temperature rise due to these pulses is not normally a concern because of their extremely short duration. However, the sum of these power losses in an application where the device is repetitively switching on and off can be significant. In applications where the phase current is changing in a sinusoidal fashion the IGBT current and duty cycle are constantly changing making loss estimation quite difficult. Switching losses are divided into IGBT and diode turn-on and turn-off loss. Switching losses in diodes are normally lower than switching losses occurring in IGBTs, it is also known that the turn-on losses in diodes are relatively small, therefore they are usually neglected [4, 14]. Assuming constant switching frequency and constant voltage Vdc across the top switch collector and the bottom switch emitter, the following equations can be used for initial loss estimation; however actual losses will depend on temperature, sinusoidal output frequency, output current ripple and other factors.

$$P_{sw-s} = \left(E_{on} + E_{off}\right) \cdot f_{sw} \cdot \frac{1}{\pi} \int_0^{\pi} i_l \cdot d\theta = \frac{2}{\pi} \left(E_{on} + E_{off}\right) \cdot I_{lpk} \cdot f_{sw}$$

$$P_{rr} = E_{rr} \cdot f_{sw} \cdot \frac{1}{\pi} \int_0^{\pi} i_l \cdot d\theta = \frac{2}{\pi} \cdot E_{rr} \cdot I_{lpk} \cdot f_{sw}$$

where Eon and Eoff are respectively the IGBT turn-on and turn-off energy per pulse and per Ampere (at the working Vdc), and Err is the diode reverse recovery energy per pulse and per Ampere.

Should be information about devices' transient energy per pulse and per Ampere not available on manufacturer's catalogue, the following expressions can be used instead of (15, 16):

$$P_{sw-s} = \frac{1}{\pi} \cdot V_{dc} \cdot I_{lpk} \cdot f_{sw} \left(t_{c-on} + t_{c-off}\right)$$

$$P_{rr} = \frac{1}{4} \cdot V_{dc} \cdot I_{rr} \cdot f_{sw} \cdot t_{rr}$$

where tc-on and tc-off are respectively the turn-on and turn-off IGBT crossover times, trr is the diode reverse recovery time, and Irr is the diode peak recovery current. In any case manufacturer's data taken at Tj=125°C should be used.

The total power losses per IGBT and per diode result respectively:

$$P_s = P_{on-s} + P_{sw-s}$$

$$P_d = P_{on-d} + P_{rr}$$

By considering the data-sheet information shown in Tab. 1, the values shown in Tab. 2 (the switch current is maximized and then φ is set to 0 and M to 1) are found for the components PM300DVA120 (Powerex-Mitsubishi) and MIG300Q101H (Toshiba) [16, 17]. Unfortunately, MIG300Q101H data-sheets are not exhaustive and most of the information are obtained by interpolating curves from lower current IPM Toshiba components (MIG150Q101H and

MIG200Q101H); as a consequence, for this component power losses calculation (and then cooling system design) is less accurate.

Table 5-2 IPM Components Data

	V_{CEn} (V)	V_{CE0}(V)	I_{Cn} (A)	V_{Fn} (V)	V_{F0} (V)	E_{on} (mJ/pulse)	E_{off} (mJ/pulse)	E_{rr} (mJ/pulse)	$t_{c\text{-}on}$ (µs)	$t_{c\text{-}off}$ (µs)	t_{rr} (µs)	I_{rr} (A)
PM300DVA120	2.75	1.1	300	2	0.6	0.1	0.2	/	0.2	0.48	0.1	200
MIG300Q101H	2.7	1.1	300	2	0.7	/	/	/	0.39	0.3	0.12	200

Table 5-3 Power Losses

	$I_{av\text{-}s}$ (A)	$I_{av\text{-}d}$ (A)	$I_{RMS\text{-}s}$ (A)	$I_{RMS\text{-}d}$ (A)	$R_{on\text{-}s}$ (mΩ)	$R_{on\text{-}d}$ (mΩ)	$P_{on\text{-}s}$ (W)	$P_{on\text{-}d}$ (W)	$P_{sw\text{-}s}$ (W)	$P_{sw\text{-}s}$ (W)	P_{rr}(W)
PM300DVA120	96.7	11.65	115.66	33	5.5	4.67	180	12	650	590	80
MIG300Q101H	96.7	11.65	115.66	33	5.3	4.3	177	12.8	/	598	96

5.3.3 Cooling System Design

On the basis of the power losses calculated as shown in the previous paragraph the phase-leg heat-sink can be designed. The design of the heat-sink for the present application is carried out by considering the equivalent thermal circuit of Fig. 5-15. The junction temperature Tj should not be higher than 125°C, whereas the ambient temperature value Ta depends on the adopted cooling system. When forced air cooling is chosen the ambient temperature is set at 30°C, in case of liquid cooled heat-sink Ta is equal to 15°C or even lower. Tc is the case temperature and for IPMs it should not be higher than 100°C in order to avoid the disabling of the device for over-temperature protection.

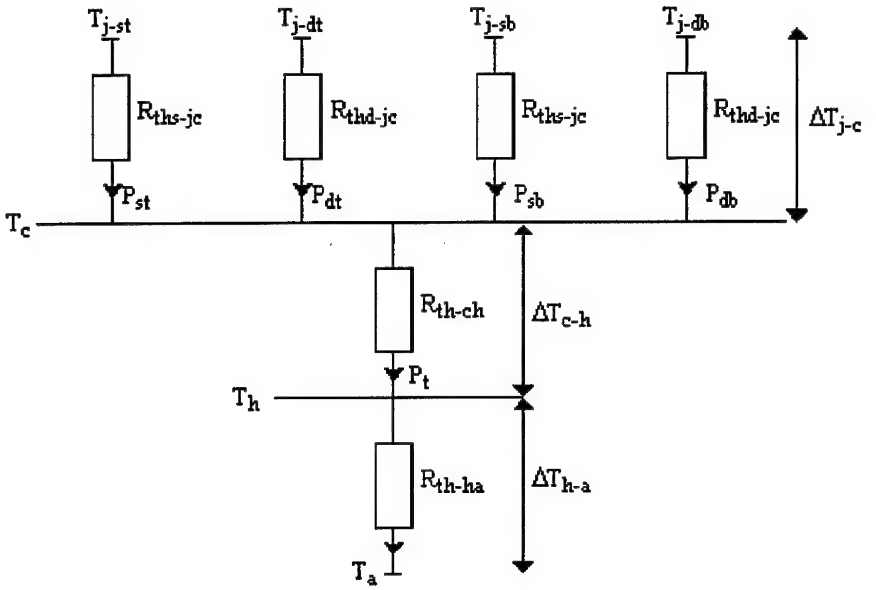


Fig. 5-15 Phase-Leg Equivalent Thermal Circuit

In Fig. 5-15 Pst=Ps, Pdb=Pd and Psb=Pdt=0 in the first half ac phase period, whereas Psb=Ps, Pdt=Pd and Pst=Pdb=0 in the second half ac phase period. Assuming that the maximum over-temperature is found on IGBTs and not on diodes, it results [7]

$$\Delta T_{j-c} = P_s\left[\frac{1}{2}R_{ths-jc} + \frac{1}{2}R'_{ths-jc} - R''_{ths-jc} + R'''_{ths-jc}\right] = P_s \cdot R^{eq}_{ths-jc}$$

where Rths-jc is the steady-state thermal resistance (0.09°C/W for PM300DVA120, 0.078°C/W for MIG300Q101H), R'ths-jc is the transient thermal impedance at 1.5Tac - Tac is the ac phase main period - (0.045°C/W for PM300DVA120, 0.03°C/W for MIG300Q101H), R''ths-jc is the transient thermal impedance at Tac (0.036°C/W for PM300DVA120, 0.022°C/W for MIG300Q101H) and R'''ths-jc is the transient thermal impedance at 0.5Tac (0.027°C/W for PM300DVA120, 0.018°C/W for MIG300Q101H). The maximum values of the data sheet thermal resistances have been considered.

From the equivalent thermal circuit of Fig. 5-15 and Table 5-3 values it is found:

	R^{eq}_{ths-jc} (°C/W)	ΔTj-c (°C)	ΔTc-h (°C)	Rth-ha (°C/W) [air cooling]	Rth-ha °C/W) [water cooling]
PM300DVA120	0.0585	45	25.9	0.028	0.045
MIG300Q101H	0.05	38.8	26.5	0.033	0.050

The forced air cooling system is designed in order to achieve Rth-ha = 0.03°C/W. The water cooling system is designed in order to achieve Rth-ha = 0.04°C/W; then comparisons between the two different cooling systems are provided focusing overall dimensions and costs.

5.3.3.1 Forced Air Cooling System

On the basis of the previously fixed specifications the following products were found for the forced air cooling system:

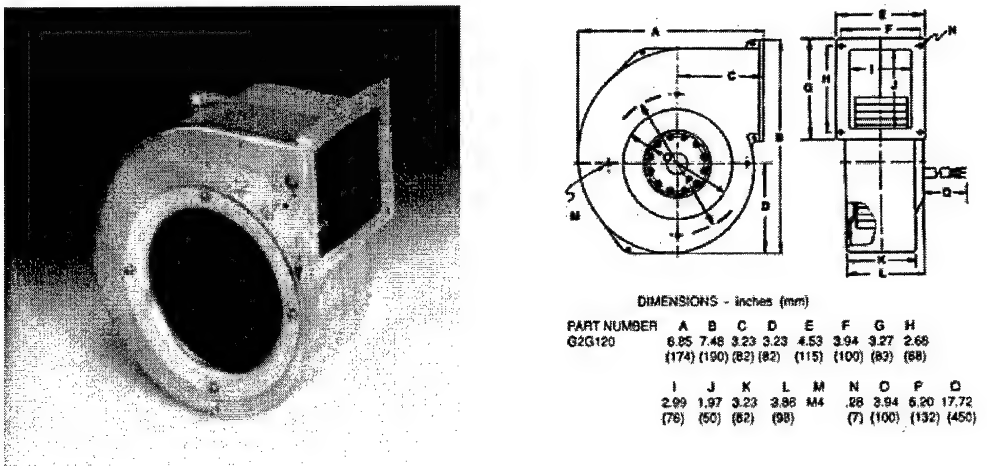
Table 5-4 Forced Convection Heat-Sink

Manufacturer	Part Number	Length (mm)	Width (mm)	Height (mm)	Air-Flow Speed (m/s)	Price ($)
AAVID [18]	4200 34 U 8 1000*	254	133.35	140	5.7	
AAVID [18]	4200 34 U 8 1200	305	133.35	140	4.5	
R-THETA [19]	MFP305T13A36AS 049D	305	127	127	4.5	
R-THETA [19]	MFP254T13A36AS 049D*	254	127	127	5.7	
R-THETA [19]	MF203T13A58AS0 49DL	203	203	64	3.5	
R-THETA [19]	AF203T13A37AS0 91DL	203	206	104	3.5	
PADA [20]	LP6D 144/200**	200	144	134	6	62
PADA [20]	LP6D 240/100**	100	240	134	6	64
WAKEFIELD [21]	BE8556 10 U 9F	254	152	140	5.7	
WAKEFIELD [21]	BE8545 08 U 9F	203	203	140	3.5	

(*) no standard product

(**) VAT included

As concern the fans to be used for supplying the cooling air flow, different options can be chosen from product catalogues. In the following are shown some examples: centrifugal blowers and axial fans seem to be the most suitable alternatives for the investigated application. The centrifugal blower and the axial oval-shape fan are characterized by the highest air-flow speed (the centrifugal blower has a small cross sectional area of the outflow air passage, whereas the axial oval-shape fan has high output volumetric flow rate), and then they can be used to provide air flow speed up to 6 m/s. However, the centrifugal blowers require, on their inflow side, enough air volume and distance from other objects in order to achieve rated performance; whereas the axial oval-shape fan has a cross sectional area which fits only the highest heat-sinks of Table 5-4. The axial square-shape fans are manufactured in several sizes and then they can be easily mounted on each heat-sink of Table 5-4; however, when the required air-flow speed is higher than 5 m/s two fans must be used in either parallel or push-pull configuration depending on the heat-sink overall dimensions.

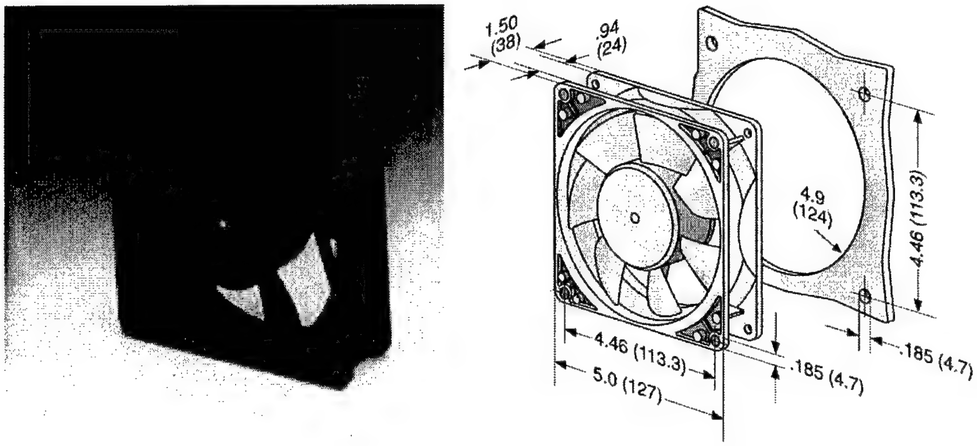


DIMENSIONS - inches (mm)

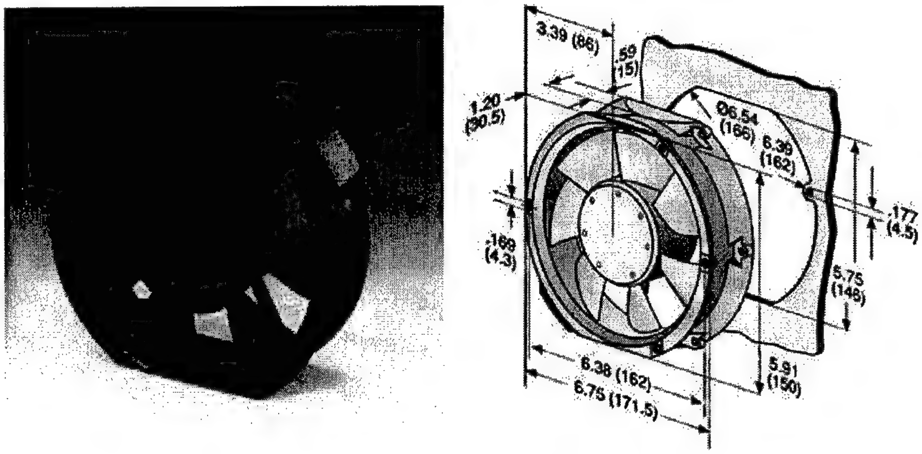
PART NUMBER	A	B	C	D	E	F	G	H
G2G120	6.85 (174)	7.48 (190)	3.23 (82)	3.23 (82)	4.53 (115)	3.94 (100)	3.27 (83)	2.68 (68)

I	J	K	L	M	N	O	P	Q
2.99 (76)	1.97 (50)	3.23 (82)	3.86 (98)	M4	.28 (7)	3.94 (100)	5.20 (132)	17.72 (450)

A) G2G085-AB04-10 Centrifugal Blower ($ 140.70)



B) 5212NH Axial Square-Shape Fan ($ 68.20)



C) 6412M Axial Oval-Shape Fan ($ 116.20)

5.3.3.2 Water Cooling System

On the basis of the power losses calculation, the found value for the thermal resistance of the liquid plate and the available product catalogues, the following specifications were established for the water heat-sink:

Length	180 mm
Width	127 mm
Max Water Flow	5.7 l/min

The following product were found:

Table 5-5 Water Cooled Heat-Sink

Manufacturer	Part Number	Length (mm)	Width (mm)	Tube OD (mm)	Water Flow Rate (l/min)	Price ($)
AAVID	418101U00000	178	197	9.5	1.5	
AAVID	416501U00000	178	127	9.4	1.5	
LYTRON	CP10G14	152	89	9.5	1.5	65
WAKEFIELD	180-12-6C	152	197	9.5	2	
WAKEFIELD	180-20-6C	152	140	9.5	2	
R-THETA	AA180TB120D2C55*	180	120	7.9	3	
R-THETA	AA152TB152D4C55*	152	152	7.9	3	
PADA**	Superplate 127x15/180**	180	127	10	3	60

(*) valves included (**) VAT included

Either parallel or series connection of the phase-leg liquid plates have been investigated. The series connection allows an easier mechanical design but a lower cooling efficiency than the parallel connection. In fact, the inlet water temperature of the last liquid plate to be cooled in a series connection is significantly higher than the temperature of the chiller outlet water; this means that in order to cool enough the last liquid plate the other ones (particularly the first liquid plate) are cooled in excess.

The parallel connection is a real flexible configuration and it allows us two options in the selection of the chiller:

1 chiller for the whole cooling system (Lytron RC030, 543x705x787 mm3, $ 5,545.00)

1 chiller for each liquid plate (Lytron RC006, 318x483x559 mm3 each, $ 2,550.00each)

In the parallel connection the liquid plates should be provided with pipe connection elements and valves in order to allow a flexible parallel connection configuration of more (3 - 4) liquid plates, thus the layout of Fig. 5-16 is obtained.

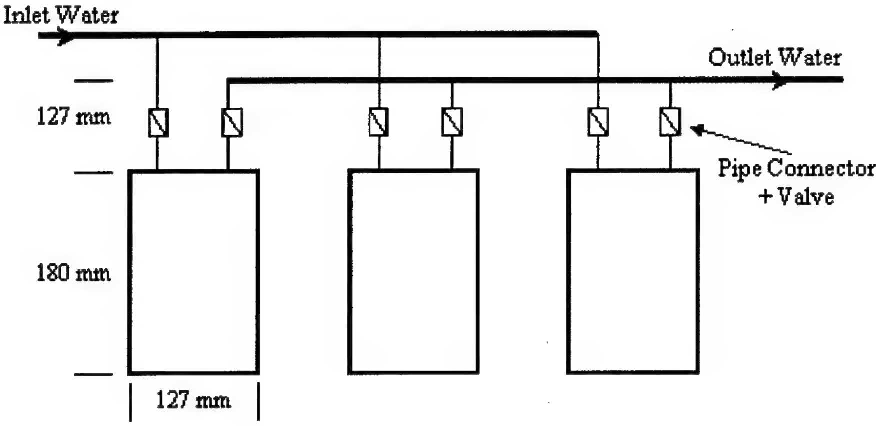


Fig. 5-16 Liquid Plates: Parallel Connection Configuration

5.3.3.3 Considerations on Cooling System Power Supply

Whichever forced air or water cooling system is chosen, power supply for fans and chiller must be provided. Fans for air cooling system can be either dc (12-24-48 V) or single phase ac (115-230 V) type; chillers for water cooler are supplied by single phase ac power net. In case of dc voltage fed fans, a dc/dc converter can provide the power directly from the phase-leg dc link;

in this case, external electrical connections for the cooling system are not required. In addition, chillers have usually power consumption higher than fans.

5.3.4 DC-Link Capacitors

In 3-phase power electronic converters the dc-link capacitor tank is sized in order to limit the dc-link voltage ripple:

$$\Delta V_{pp} = \frac{\sqrt{3} \cdot (1-M) \cdot M \cdot I_{Lrms}}{\sqrt{2} \cdot f_s \cdot C}$$

where ΔVpp is the peak-to-peak dc-link voltage ripple, Ilrms is the rms value of the ac-phase current, fs is the switching frequency, C is the total capacitance of the tank and M is the modulation index (defined as $M = V_{llpk} / Vdc$, Vllpk is the peak value of the line-to-line voltage and Vdc is the dc-link voltage value).

However, the rms value of the ripple current in the dc-link capacitor must be also considered during the design:

$$I_{Crms} = I_{Lrms} \cdot \sqrt{M\left[\frac{1}{\pi} + \cos^2 \varphi \cdot \left(\frac{4}{\pi} - \frac{3}{2} \cdot M\right)\right]}$$

where Icrms is the ripple current in the capacitor tank and cosφ is the displacement power factor.

Eq. (1) and (2) reach their maximum value for M=0.5 and cosφ=1. For Vdc=800V, Ilrms=125A, fs=20kHz, ΔVpp=1%Vdc it is found C=240uF, Icrms=81A.

Simulations in Saber have been carried on to verify the found theoretical values.

3-Phase PFC

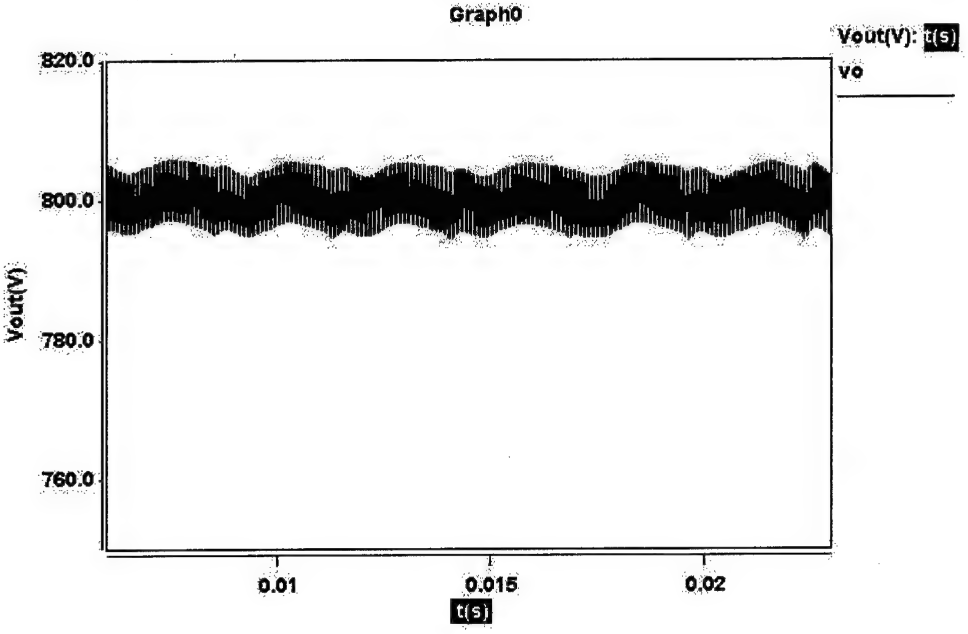


Fig. 5-17 DC-Link Voltage

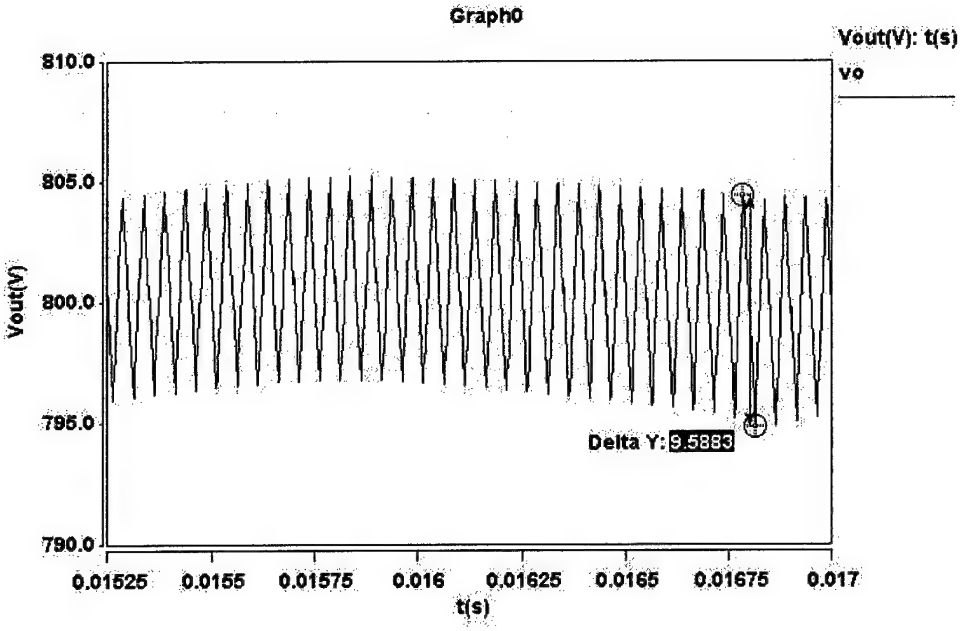


Fig. 5-18 DC-Link Voltage: zoom

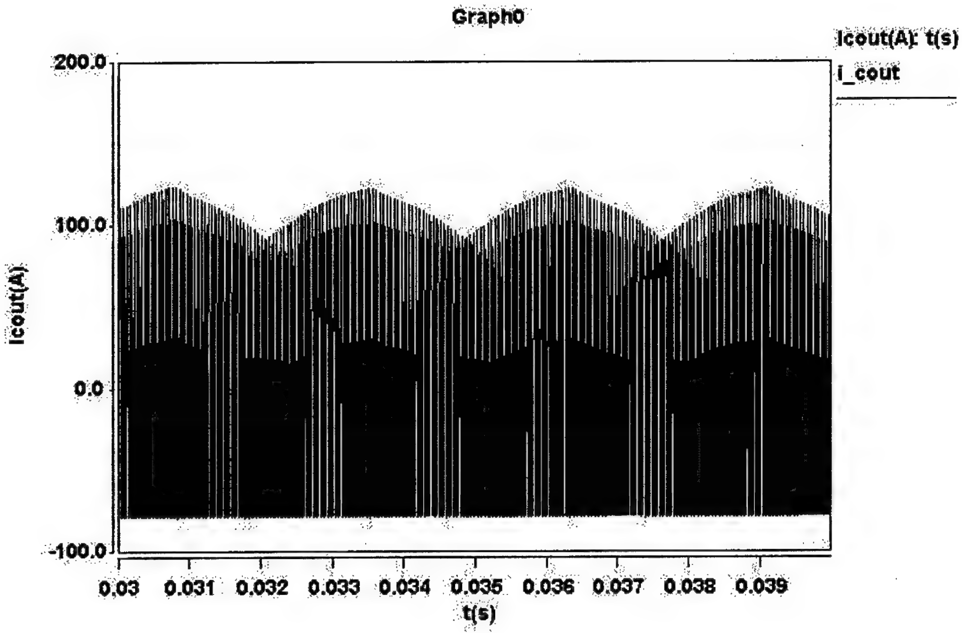


Fig. 5-19 DC-Link Capacitor Current

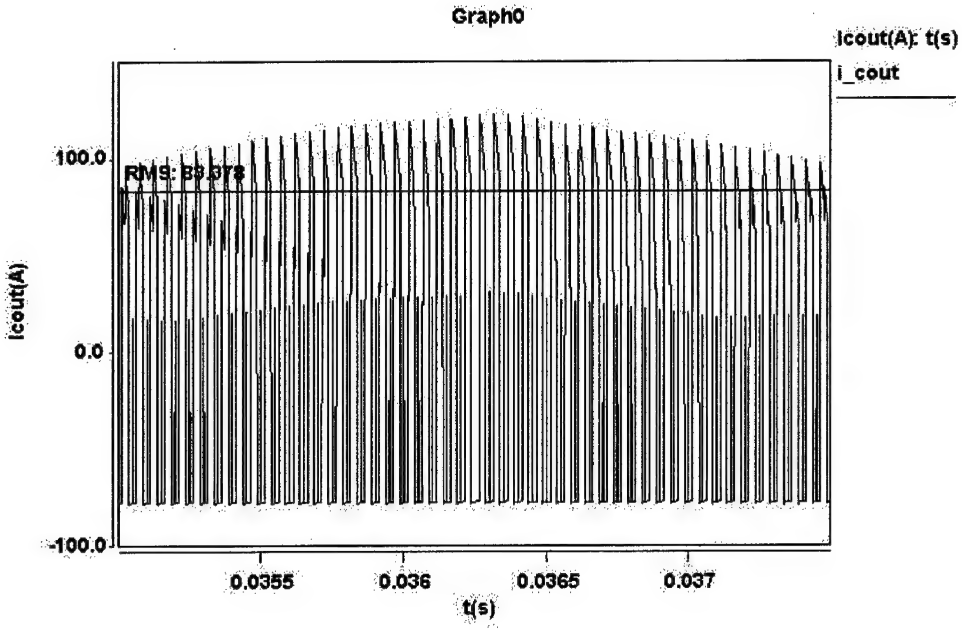


Fig. 5-20DC-Link Capacitor Current: zoom

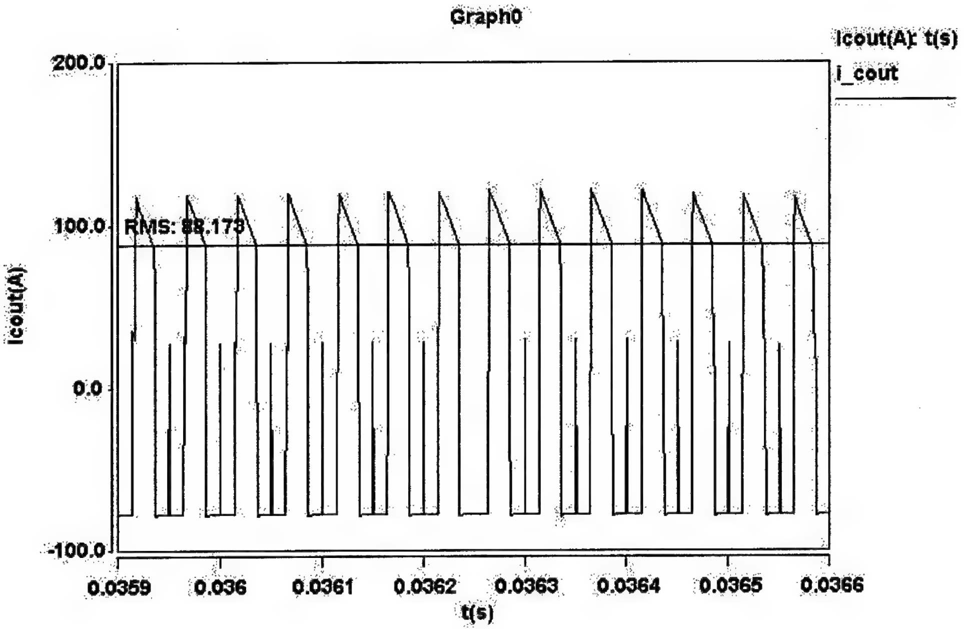


Fig. 5-21 DC-Link Capacitor Current: zoom

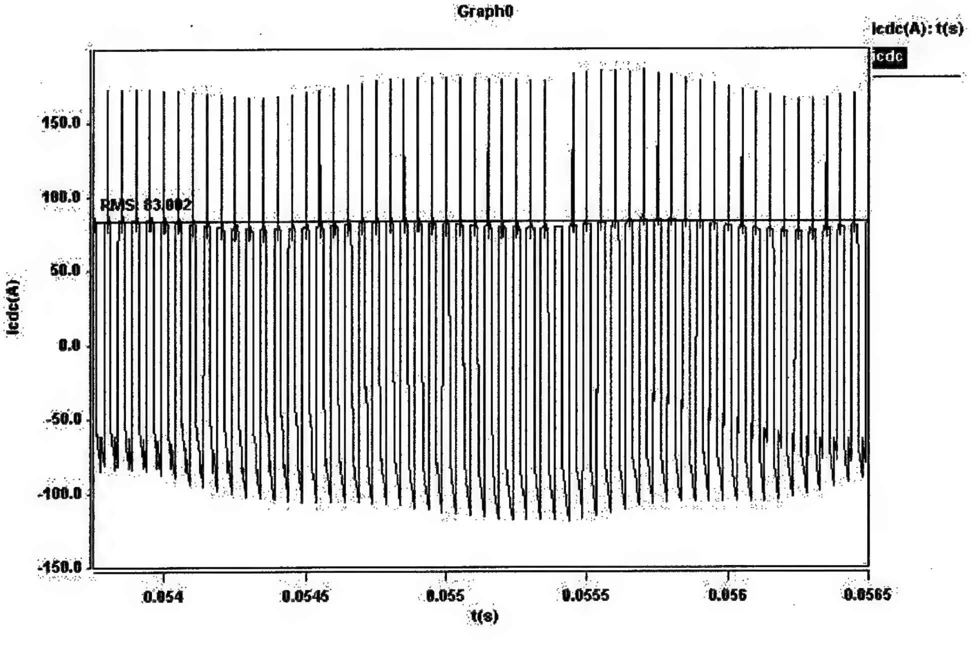


Fig. 5-22 DC-Link Capacitor Current: zoom

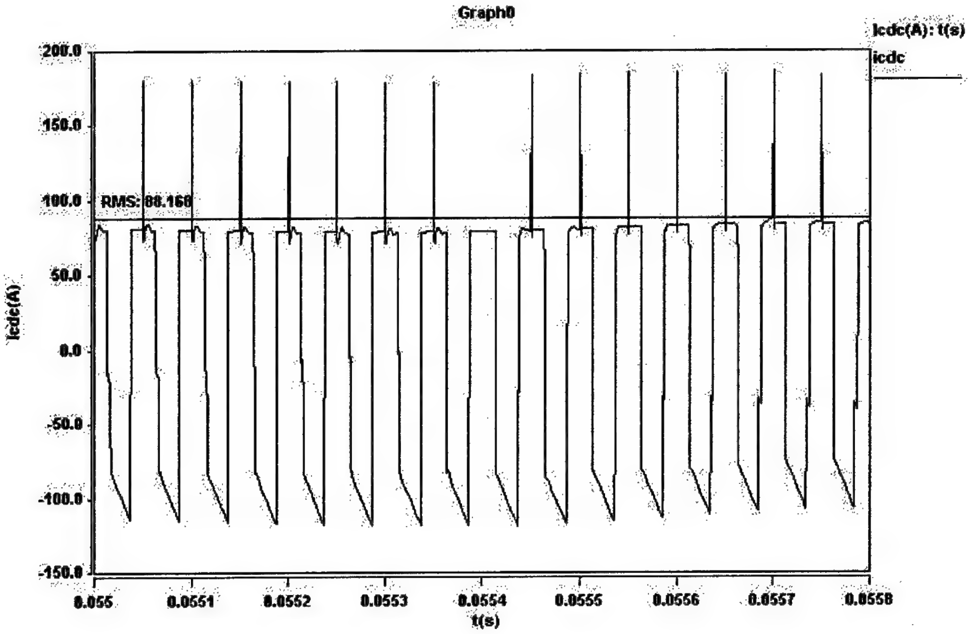


Fig. 5-23 DC-Link Capacitor Current: zoom

5.3.5 DC-Link Bus Structure

As shown in Fig. 5-10, the first version of the dc-link bus structure employs busbars to implement the positive and negative rails. This approach however has proven to be inadequate and shall require significanat modifications in the future. This work will be part of the recently started new project. As shows Fig. 5-24, the back of the cabinet consists of a series of busbars that transverse it longitudinally. The distributed structure though of the PEBBs increases significantly the parasitic effects in the converter, hence parasitic inductances are high enough that they effectively diminish the capacitance of the DC bus. This negative effect was reduced by distributing the dc-link capacitors along the DC busbars, thus eliminating the actual capacitor bank by placing the caps directly behind each PEBB connector.

In the future however the parasitics problems not addressed before will be furthered and studied. In fact it has already been determined that a planar structure where the positive and negative busbars actually form a now parasitic capacitance should be the ideal approach and solution. These busbars are closely positioned and properly isolated using Kapton. Future work is required though to effectively design these new dc-link planes.

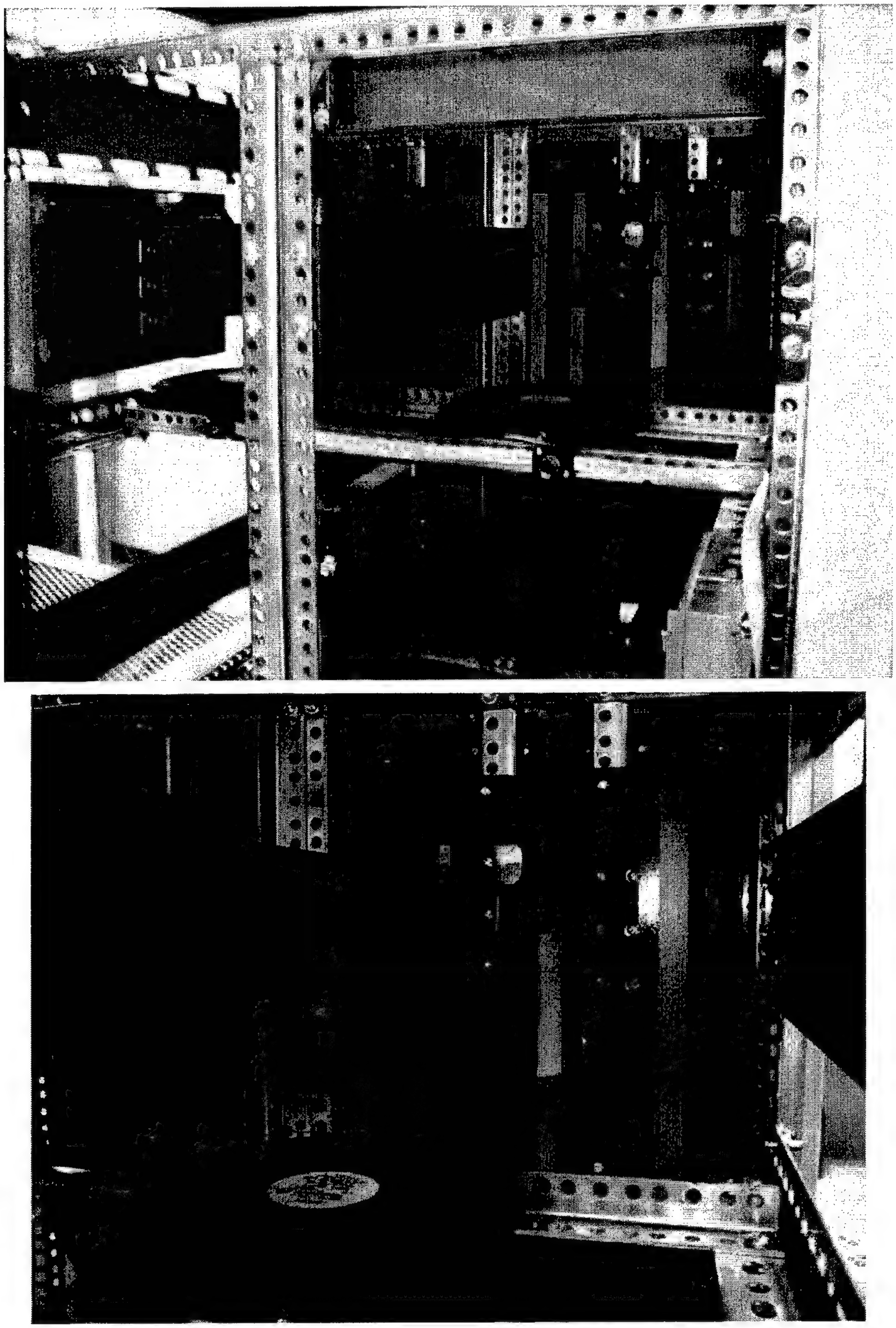


Fig. 5-24 PEBB, sliding rails, dc-link connectors and busbars of the cabinet under construction.

5.3.6 AC Inductors and Capacitors

In three-phase power electronic converters the ac-phase inductor is sized in order to limit the ac current ripple:

$$\Delta I_{Lpp} = \frac{2 \cdot V_{dc}}{3 \cdot L \cdot f_s} \cdot (1 - M) \cdot M$$

where ΔIlpp is the peak-to-peak ac current ripple and L is the inductance value of the phase inductor.

The ac-phase capacitor is sized in order to limit the ac-phase voltage ripple:

$$\Delta V_{Lpp} = \frac{\Delta I_{Lpp}}{4 \cdot f_s \cdot C_{ac}}$$

where ΔVlpp is the ac-phase voltage ripple and Cac is the capacitance value of the phase capacitor.

For ΔIlpp=20%Ilpk (34A) and ΔVlpp=2%Vl (5.54V) it is found L=186uH, Cac=75uF

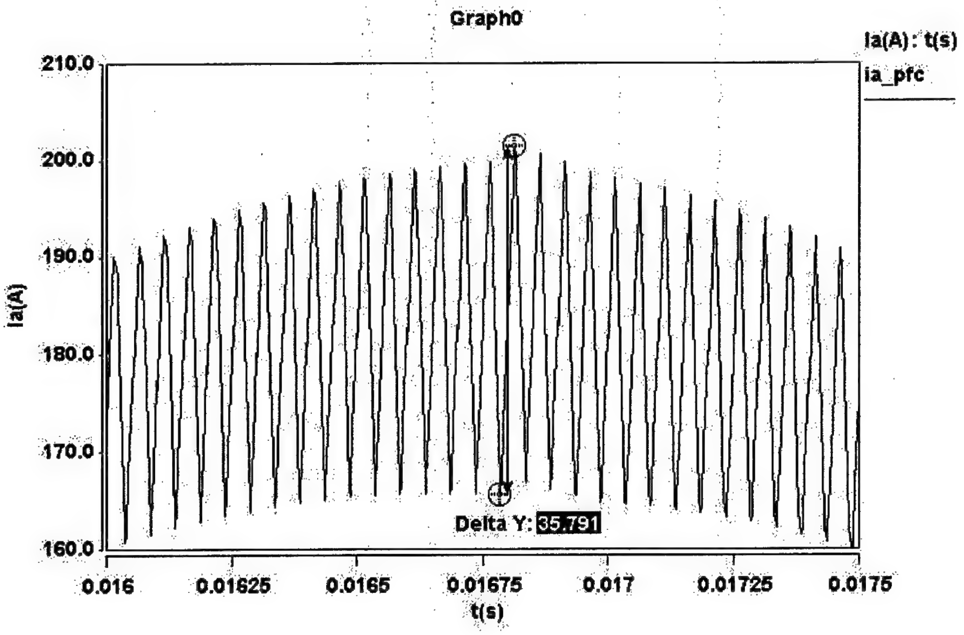


Fig. 5-25 AC-Phase Current: zoom

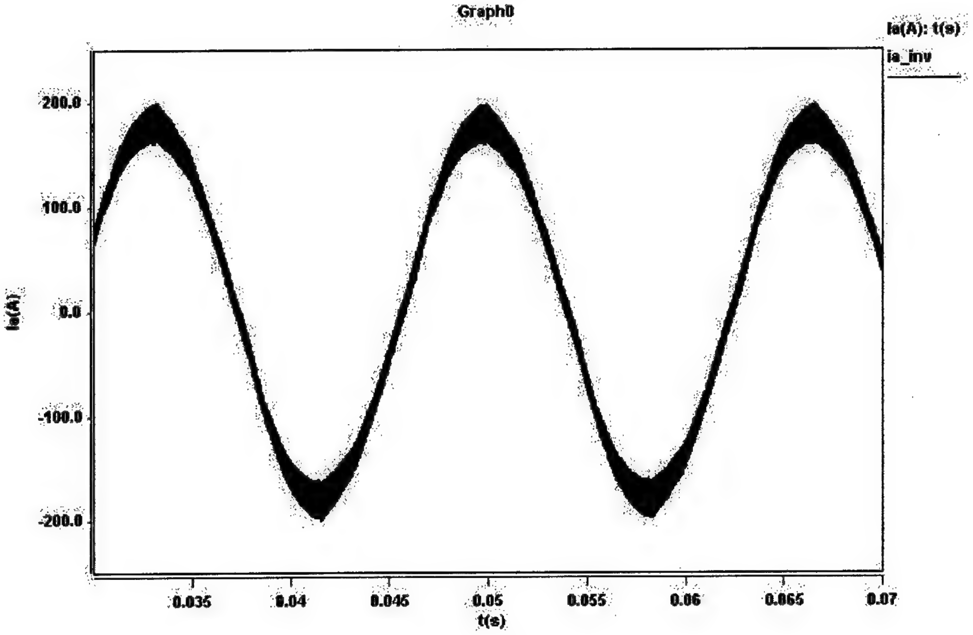


Fig. 5-26 AC-Phase Current

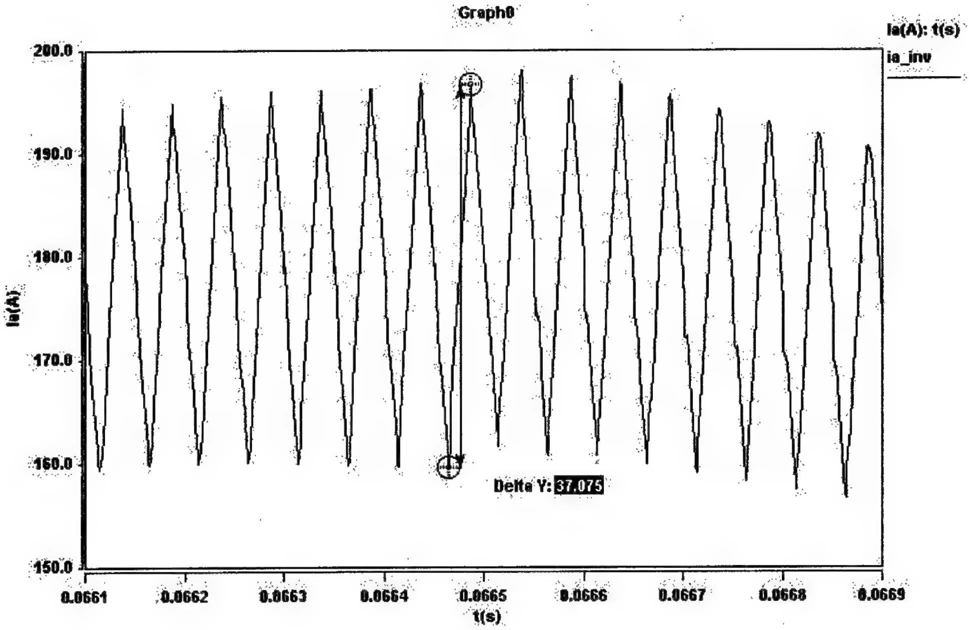


Fig. 5-27 AC-Phase Current: zoom

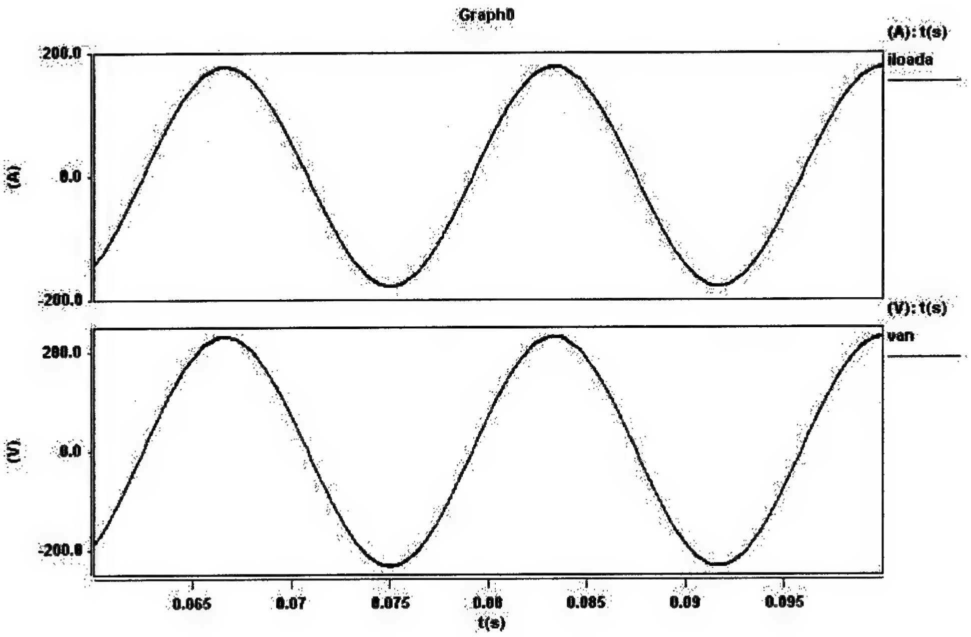


Fig. 5-28 Load-Phase Current (top) and Voltage (bottom)

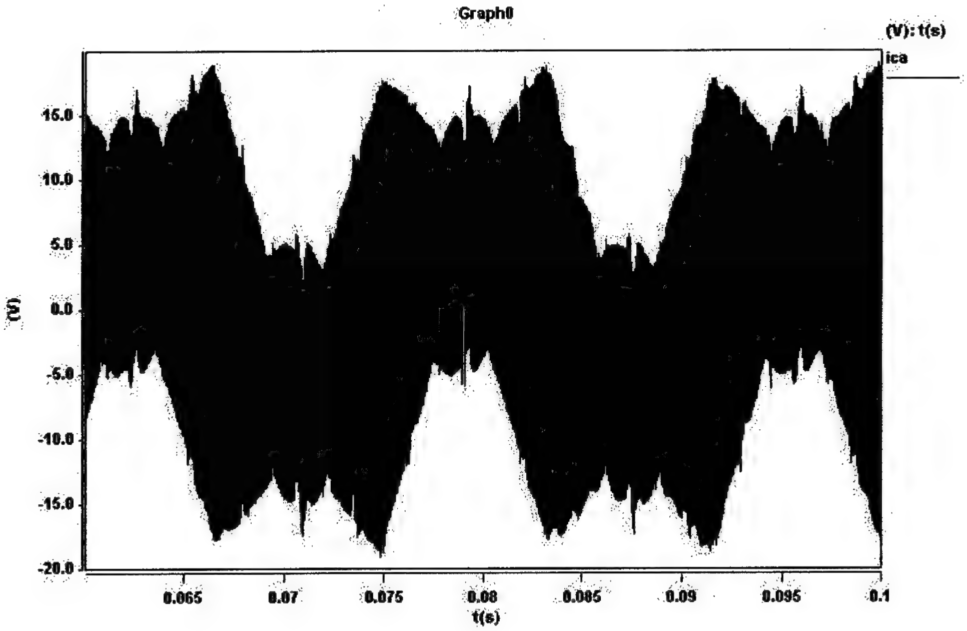


Fig. 5-29 AC-Phase Capacitor Current

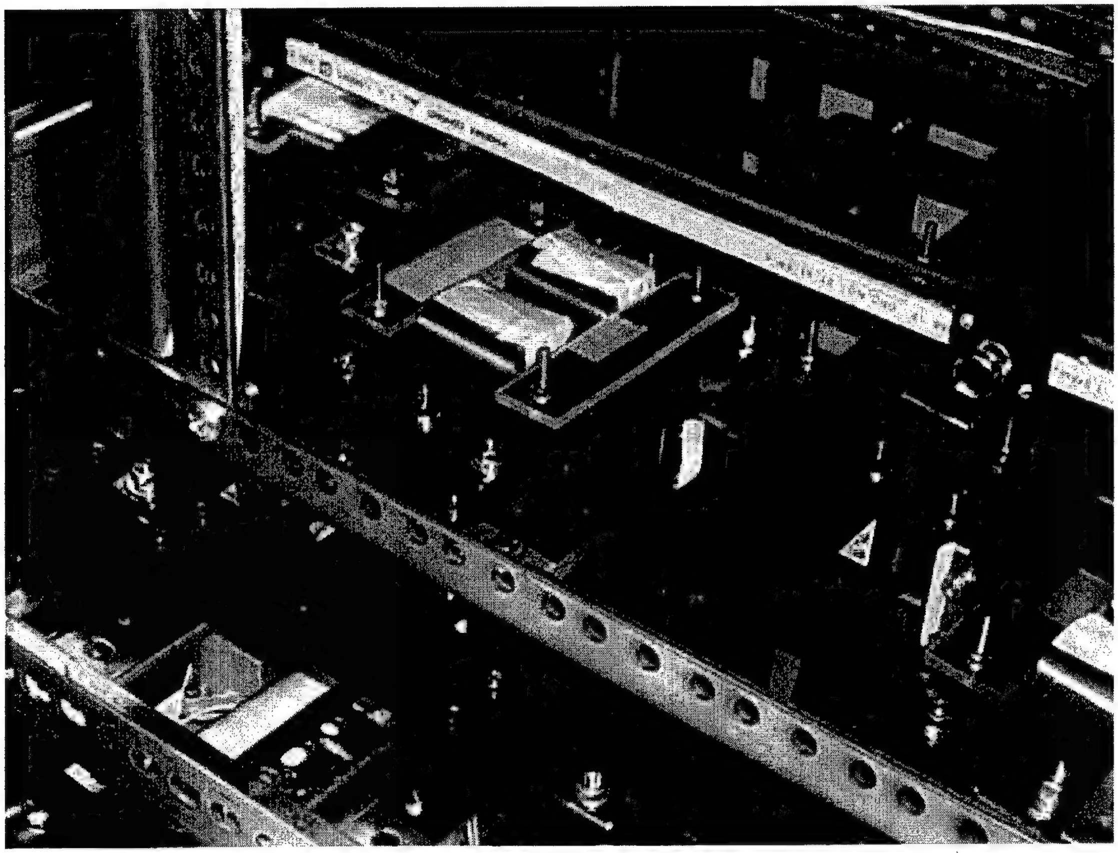


Fig. 5-30 Closeup view of the AC inductors positioned beneath the fans in the cabinet.

5.3.6.1 Inductor Design

Design input parameters:

inductance value L;

rated peak current (ILpk) and rated rms current (ILrms);

operating frequency (fs);

In order to carry on the design of the inductor the following expressions are considered:

$$V_e = A_e \cdot l_e = \frac{\mu_0 \mu \cdot L \cdot I_{Lpk}^2}{B_{pk}^2}$$

$$N = \frac{L \cdot I_{Lpk}}{A_e \cdot B_{pk}}$$

$$L = \frac{\mu_0 \mu \cdot A_e \cdot N^2}{l_e}$$

$$A_w = \frac{L \cdot I_{Lpk} \cdot I_{Lrms}}{k_{cu} \cdot J_{Lrms} \cdot A_e \cdot B_{pk}}$$

where Ve, Ae, le and µ0·µ are respectively the volume, the cross section, the path length and the magnetic permeability of the material in which the magnetic energy is stored; Bpk is the peak flux density in the core; Aw, N, kcu and JLrms are respectively the required winding area, the number of turns, the fill factor and the current density considered for the inductor winding.

The core material is chosen on the basis of the operating frequency; thus, the value of Bpk and the magnetic permeability are found from data-sheet of the selected material. The core size closest to the result is chosen among the options available on the market; then, by using the previous equations the actual values of Bpk, N, L and Aw are known. For the current design Metglass material is chosen and the following values are used in the previous equations:

L=185uH, ILrms=125A, JLrms=4A/mm2, Bpk=1.2T, kcu=0.5

Selecting the core AMCC-400 it is found:

Ae=11.7cm2, le=5.4mm, N=26, L=184uH, Aw=16cm2

The required copper foil section is

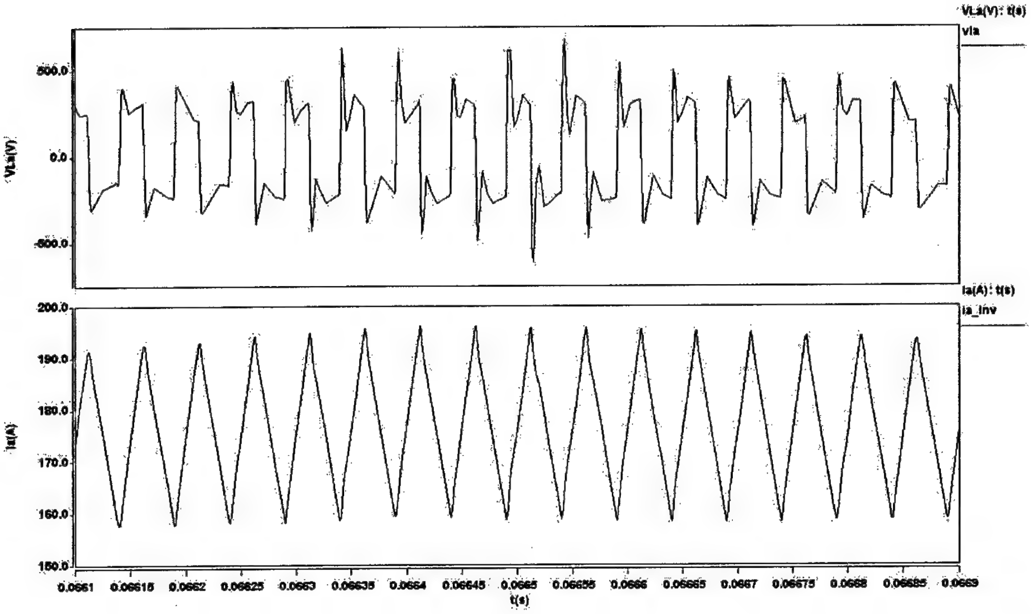
Scu=62.5mmx0.5mm.

5.3.6.2 Selected Capacitors

dc-link capacitor for each phase-leg: electronic concepts UL31BL356K, 1000V-35uF;

additional dc-link capacitor: electronic concepts UL30BL0085, 1000V-85uF;

ac-phase capacitor: electronic concepts 5MPA2606J, 530VAC-60uF;



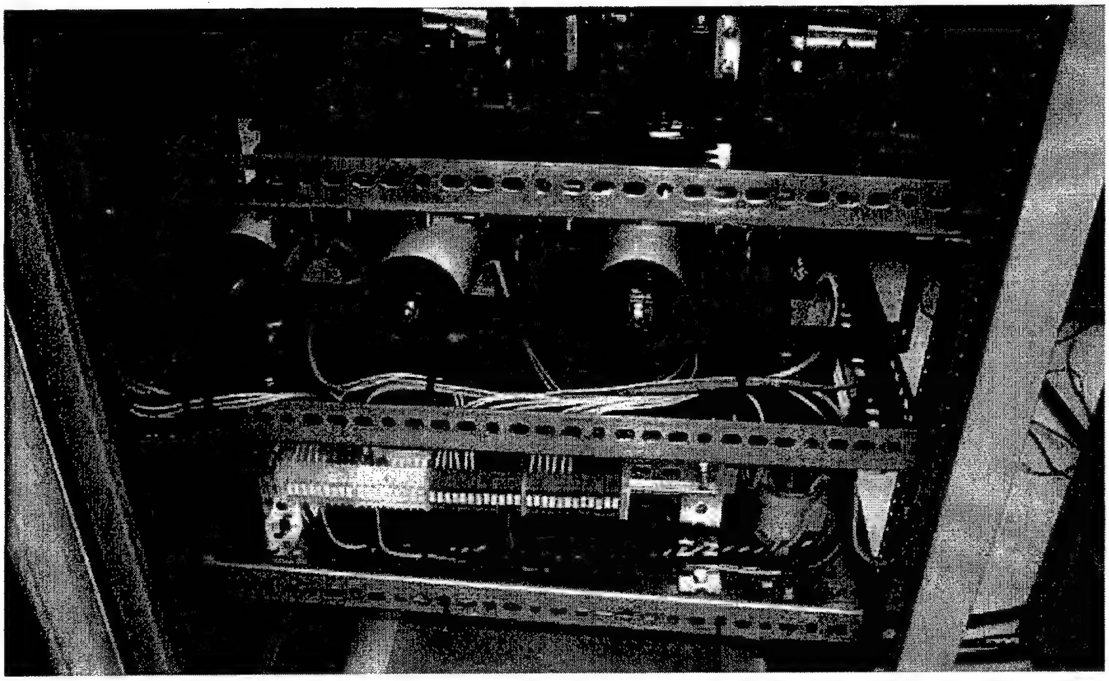


Fig. 5-31 AC capacitors view inside the cabinet. These are directly positioned and connected to the output terminals of the AC inductors.

5.3.7 *Contactors, fuses, and power connectors*

The contactors and fuses were placed according to the scheme shown in Fig. 5-32, a view of the final mounting in the cabinet is shown in Fig. 5-33. The high power connectors were placed as shown in Fig. 5-34.

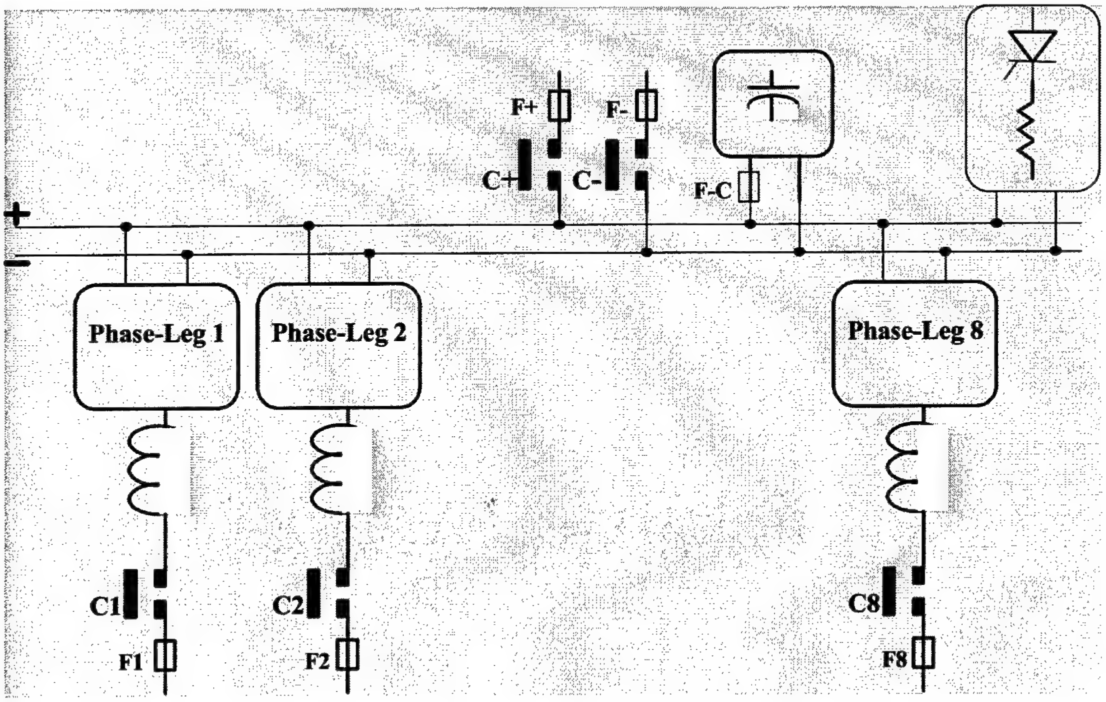


Fig. 5-32 Contactors and fuses scheme for the experimental prototype.

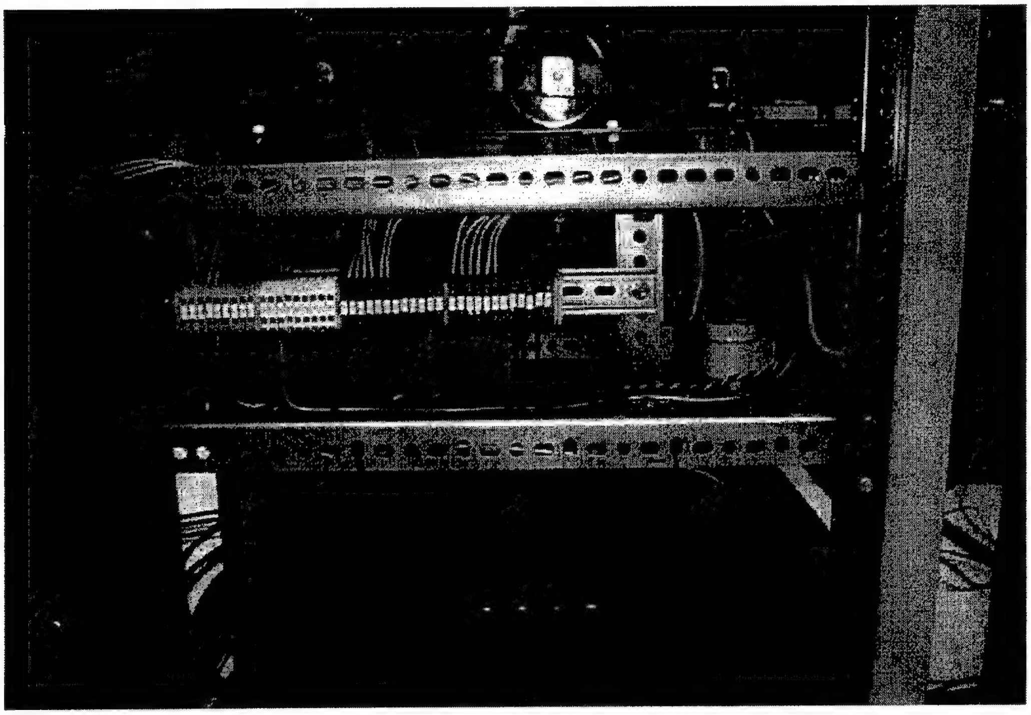


Fig. 5-33 Closeup view of the contactors, fuses and control power connectors.

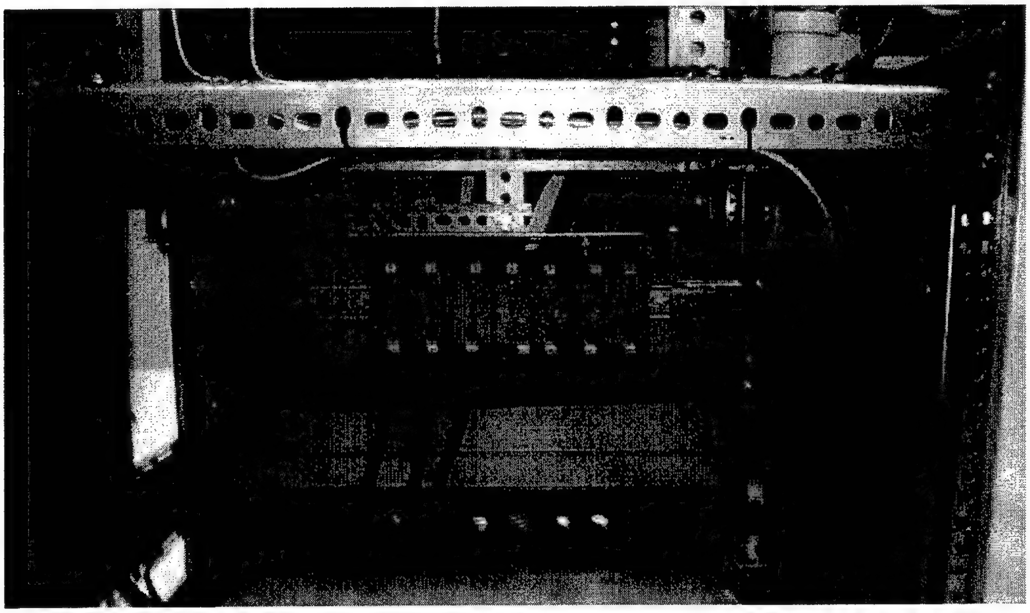


Fig. 5-34 View of the power terminal blocks at the bottom of the cabinet structure.

5.3.8 Investigation on Soft-Switching Circuit Topology

It is well known that the zero-current-transition (ZCT) technique is attractive in high-power inverters and power-factor-correction (PFC) rectifiers, where the minority-carrier devices, such as IGBTs, are the power switches. The basic concept of ZCT technique is to force the current of an outgoing device in PWM converters to zero prior to turning off the device. By using the ZCT technique, converters can achieve a higher switching frequency with reduced switching losses and less electro-magnetic-interference (EMI).

Generally the ZCT commutation is realized through the oscillatory action of an LC circuit, which is triggered by an auxiliary switch. In power electronics building block (PEBB) phase-leg configuration, it is preferred to assist each main switch independently so that any PWM schemes for the hard-switching counterparts can be directly employed without modification and it is also easy to assemble more PEBB phase-legs in order to arrange several converter topologies.

Using the same ZCT concept, a variety of circuit configurations and control schemes can result in different operational behaviors and soft-switching features. Fig. 1 shows one ZCT PEBB phase-leg, which consists of two main switches (S1 and S2), two auxiliary switches (S1x and S2x) and one LC resonant tank. Based on the same circuit configuration, a number of ZCT control schemes have been developed for modern gate-controlled devices, such as the IGBTs. These include the following: the ZCT scheme which achieves zero-current turn-off for the main switches [xx]; the improved scheme which also provides soft commutation for the main diodes and the auxiliary switches [xxi]; and the further improved scheme which achieves near-zero-voltage turn-on for the main switches [xxii]. The last introduced scheme (ZCT-NZVT) seems to the most promising one from the point of view of the resonant capacitor voltage stress, the auxiliary switch stress distribution, the peak value of both the current in the main switch and the current circulating in the auxiliary switch and also from the point of view of the circulating energy in the resonant tank [xxiii].

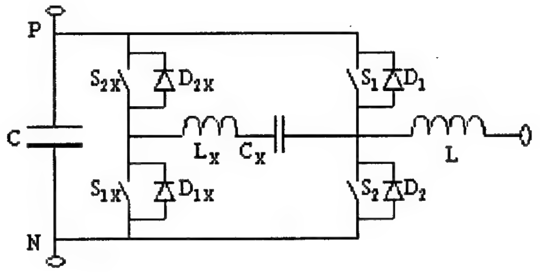


Fig. 5-35 Soft switching PEBB schematic

Further, two attributes can be identified in the circuit shown in Fig. 5-35. First, soft transitions for each phase-leg are executed independently. Hence, any PWM scheme developed for hard-switching converters is still applicable, and the ZCT implementation does not compromise any well-proven control techniques. Second, voltage stresses across all devices are kept to the level of dc bus voltage because no additional devices or components are inserted in the main power path.

In the following the design steps for both ZCY-NZVT and IZCT techniques are resumed on the basis of the normalized approach. The normalization factors are the PEBB maximum dc link voltage Vdcm and the maximum current Ilm subjected to the ZCT turn-off; thus the normalized expressions are:

$$I_{\ln} = I_l / I_{lm}\ ; \quad V_{dcn} = V_{dc} / V_{dcm}\ ; \quad Z_{0n} = Z_0 \cdot \frac{I_{lm}}{V_{dcm}}\ ;$$

where Z0 is the resonant tank impedance.

Some design expressions are common to all the ZCT techniques, thus can be provided before the analysis of each technique:

$$T_0 = 2\pi\sqrt{L_x \cdot C_x} \qquad (1)$$

$$Z_0 = \sqrt{L_x / C_x} \qquad (2)$$

$$L_x = \frac{Z_0 \cdot T_0}{2\pi} \qquad (3)$$

$$C_x = \frac{L_x}{Z_0^2} \qquad (4)$$

$$I_{xpkn} = k \cdot I_{\ln} \qquad (5)$$

$$I_{xpkoffn} = k_{off} \cdot I_{\ln} \qquad (6)$$

where Lx and Cx are the passive elements of the resonant tank, and Ixpkn and Ixpkoffn are the normalized values of respectively the peak resonant current and the peak resonant current during the main switch turning off.

5.3.8.1 ZCT-NZVT PEBB

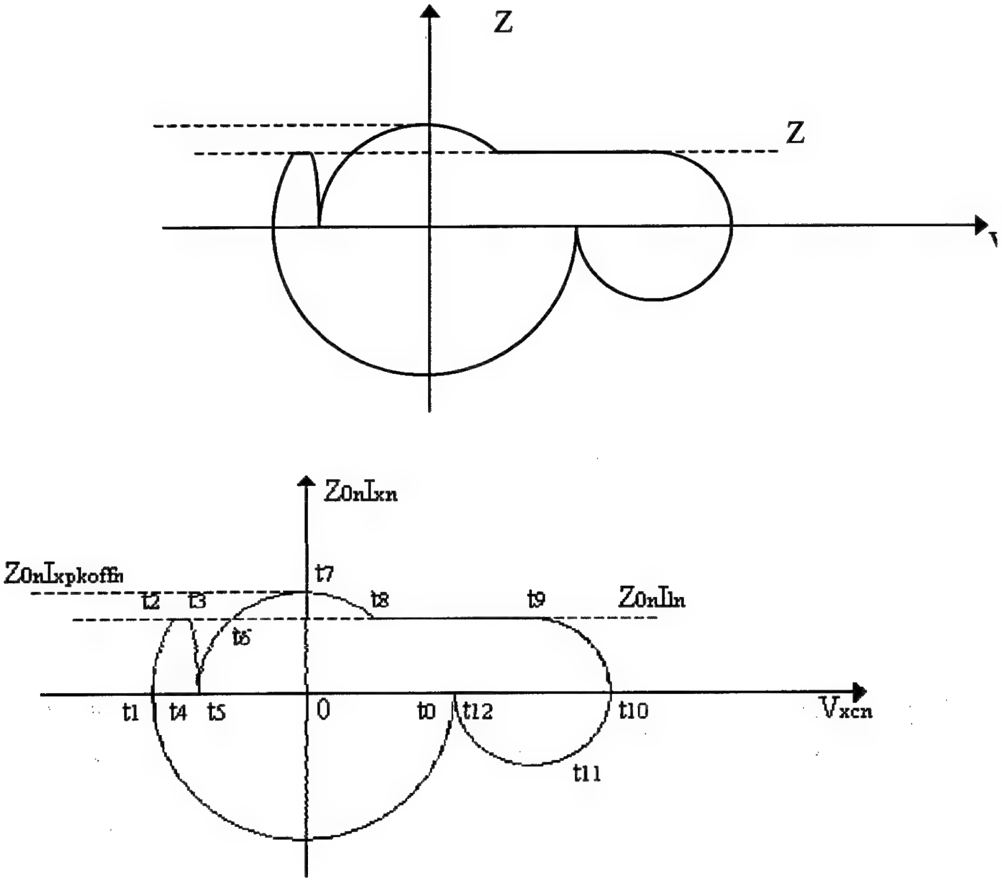


Fig. 5-36 State-Plane Trajectory

5.3.8.1.1 Design rule

Determining Z0n

From the state-plane trajectory it can be seen that

$$I_{xpkoffn} = \frac{|V_{cxn}(t_5)|}{Z_{0n}} \tag{7}$$

where Vcxn is the normalized value of the resonant capacitor voltage.

Assuming as first rough calculation that

$$|V_{cxn}(t_5)| = |V_{cxn}(t_1)| - Z_{0n} \cdot I_{\ln} = V_{cxn}(t_0) - Z_{0n} \cdot I_{\ln}$$

and knowing that

$$V_{cxn}(t_0) = V_{dcn} - Z_{0n} \cdot I_{\ln}$$

we have

$$|V_{cxn}(t_5)| = V_{dcn} - 2 \cdot Z_{0n} \cdot I_{\ln} . \qquad (8)$$

Using (6) and (7) it is found

$$Z_{0n} = \frac{V_{dcn}}{I_{\ln}} \cdot \frac{1}{2 + k_{off}} . \qquad (9)$$

It must be verified that $k_{off} \geq 1$ in order to achieve a total ZCT turn-off; thus at maximum dc link voltage Vdcm and maximum current Ilm condition ZCT turn-off is accomplished when $Z_{0n} \leq 0.33$.

As Z0n is selected, the value of koff, and then Ixpkoffn, can be found from (9) for every current and voltage operating condition. Also, the values of both resonant capacitor peak voltage (Vcxpkn) and resonant peak current (Ixpkn) can be found for each operating condition:

$$V_{cxpkn} = V_{dcn} + Z_{0n} \cdot I_{\ln} \qquad (10)$$

$$I_{xpkn} = \frac{V_{cxn}(t_0)}{Z_{0n}} = \frac{V_{dcn}}{Z_{0n}} - I_{\ln} . \qquad (11)$$

Using (5), (9) and (11) the value of k is easily found

$$k = k_{off} + 1 . \qquad (12)$$

The maximum normalized value of the peak current that flows through each power devices is respectively:

$$I_{Spkn} = I_{\ln} ; \qquad I_{Dpkn} = I_{\ln} \cdot (1+k) ; \qquad I_{SXpkn} = k \cdot I_{\ln} ; \qquad I_{DXpkn} = I_{\ln}$$

From the state-plane trajectory it is possible to get a more accurate calculation of the Vcxn(t5) value. However, the resulting expression is more difficult to manage than (8). In the following is shown the detailed calculation:

$$V_{cxn}(t_5) = V_{cxn}(t_0) - \Delta V_{cxn}(t_{4-1})$$

$$\Delta V_{cxn}(t_{4-1}) = \Delta V_{cxn}(t_{2-1}) + \Delta V_{cxn}(t_{3-2}) + \Delta V_{cxn}(t_{4-3})$$

$$\Delta V_{cxn}(t_{2-1}) = V_{cxn}(t_0) - \sqrt{V_{dcn} - 2 \cdot Z_{0n} \cdot I_{\text{ln}}}$$

$$\Delta V_{cxn}(t_{3-2}) = \sqrt{V_{dcn} - 2 \cdot Z_{0n} \cdot I_{\text{ln}}} + \sqrt{(V_{dcn} - k_{off} \cdot Z_{0n} \cdot I_{\text{ln}})^2 - (Z_{0n} \cdot I_{\text{ln}})^2} - V_{dcn}$$

In the period $\Delta t = t_4 - t_3$ the Vdc voltage is included in the resonant path and as result the resonant current decreases rapidly toward zero; thus we can assume, with a good approximation, that the capacitor Cx is charged by means the current Ix linearly decreasing from Il to zero. The following expressions can be written:

$$\Delta V_{cx}(t_{4-3}) = \frac{I_l \cdot \Delta t_{4-3}}{2 \cdot C_x}$$

$$\frac{L_x \cdot I_l^2}{2} = \frac{C_x \cdot V_{cx}^2(t_{4-3})}{2} + V_{dc} \cdot I_l \cdot \Delta t_{4-3}.$$

By rearranging the above written equations, the variation of the resonant capacitor normalized voltage level during the period Δt4-3 results:

$$\Delta V_{cxn}(t_{4-3}) = -2 \cdot V_{dcn} + \sqrt{4 \cdot V_{dcn}^2 + Z_{0n}^2 \cdot I_{\text{ln}}^2}$$

Finally it is found an accurate expression for Vcxn(t5):

$$V_{cxn}(t_5) = 3 \cdot V_{dcn} - \sqrt{(V_{dcn} - k_{off} \cdot Z_{0n} \cdot I_{\text{ln}})^2 - (Z_{0n} \cdot I_{\text{ln}})^2} - \sqrt{4 \cdot V_{dcn} + (Z_{0n} \cdot I_{\text{ln}})^2}$$

5.3.8.1.2 Choosing T0

From the state-plane trajectory, in case of Ilm and Vdcm, can be assumed

$$T_{off} = 2\sqrt{L_x \cdot C_x} \cdot \cos^{-1}(1/k_{off}) = \frac{T_0}{\pi} \cdot \cos^{-1}(1/k_{off})$$

where the choice of Toff is device dependent, and it should be longer than the current fall time of the main switch.

5.3.8.1.3 Calculation of Lx and Cx

The values of Lx and Cx can be determined by means of respectively (3) and (4)

5.3.8.1.4 Timing of auxiliary switch gating signal

Assuming that the main leg active components are S1 and D2, the switch S1X is gated to achieve turning-on of the main switch S1 whereas S2X provides the turning-off of S1.

The leading time of gating S1X with respect to S1 can be set as

$$T_{S1X} = \frac{T_0}{2} \cdot \left(1 + \frac{1}{2k}\right)$$

The pulse width of S1X gate signal is $PW_{S1X} = T_{S1X}$ in order to have S1X turn-off and S1 turn-on at the same time.

The leading time of gating S2X with respect to S1 turn-off signal can be chosen as

$$T_{S2X} = \frac{T_0}{4} \cdot I_{\mathrm{In}}$$

whereas the pulse width of S2X gate is pointed out as

$$PW_{S2X} = \frac{T_0}{2} + \Delta T \cdot \left(2 - I_{\mathrm{In}}\right) .$$

In the previous expression

$$\Delta T = \frac{\Delta T_{off}}{\Delta V_{off}} \cdot \left[V_{cxpk} - V_{cx}(t_8)\right] ,$$

where

$$\Delta T_{off} = t_9 - t_8 = C_x \cdot \frac{\Delta V_{off}}{I_l} ;$$

$$\Delta V_{off} = V_{dc} - Z_0 \cdot I_l \cdot \sqrt{k_{off}^2 - 1} ;$$

$$V_{cx}(t_8) = Z_0 \cdot I_l \cdot \sqrt{k_{off}^2 - 1} .$$

5.3.8.2 IZCT PEBB

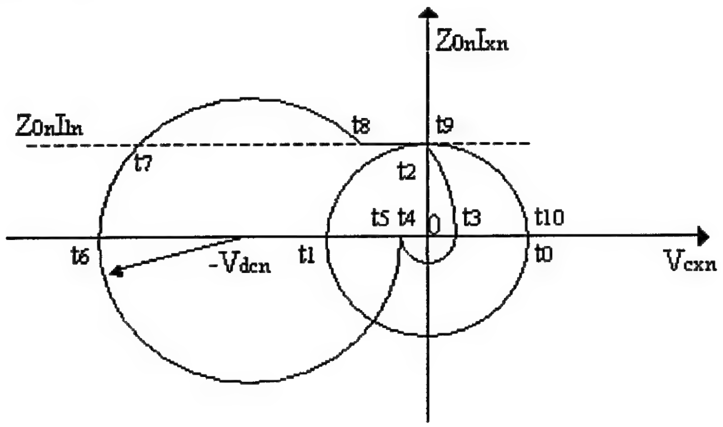


Fig. 5-37 State-Plane Trajectory

5.3.8.2.1 Design rule

Determining Z0n

From the state-plane trajectory it can be seen that

$$I_{xpkoffn} = \frac{V_{dcn} - |V_{cxn}(t_5)|}{Z_{0n}}$$

with $|V_{cxn}(t_5)| = |V_{cxn}(t_3)|$.

In the period $\Delta t = t_3 - t_2$ the Vdc voltage is included in the resonant path and as result the resonant current decreases rapidly toward zero; thus we can assume, with a good approximation, that the capacitor Cx is charged by means the current Ix linearly decreasing from Il to zero. The following expressions can be written:

$$V_{cx}(t_3) = \frac{I_l \cdot \Delta t}{2 \cdot C_x}$$

$$\frac{L_x \cdot I_l^2}{2} = \frac{C_x \cdot V_{cx}^2(t_3)}{2} + V_{dc} \cdot I_l \cdot \Delta t$$

By rearranging the above written equations, the resonant capacitor normalized voltage level at t3 results:

$$V_{cxn}(t_3) = -2 \cdot V_{dcn} + \sqrt{4 \cdot V_{dcn}^2 + Z_{0n}^2 \cdot I_{\ln}^2}$$

From the previous equations it is found that:

$$k_{off} = \frac{3 \cdot V_{dcn}}{Z_{0n} \cdot I_{\ln}} - \sqrt{\frac{4 \cdot V_{dcn}^2}{Z_{0n}^2 \cdot I_{\ln}^2} + 1},$$

and in a more promptly form

$$k_{off} = 3 \cdot k_{id} - \sqrt{4 \cdot k_{id}^2 + 1}$$

where

$$k_{id} = \frac{V_{dcn}}{Z_{0n} \cdot I_{\ln}}$$

is the ratio between Ixpkoffn and Iln when Vcxn(t5)=0.

The value of Z0n is determined by means of (16) and (17) in order to achieve $k_{off} \geq 1$, thus providing a total ZCT turn-off.

As Z0n is selected, the value of koff, and then Ixpkoffn, can be found from (16) for every current and voltage operating condition. Also, the values of the resonant capacitor peak voltage (Vcxpkn) can be found for each operating condition:

$$V_{cxpkn} = V_{dcn} + Z_{0n} \cdot I_{xpkoffn}$$

which, rearranged leads to the expression

$$V_{cxpkn} = 4 \cdot V_{dcn} - \sqrt{4 \cdot V_{dcn}^2 + Z_{0n}^2 \cdot I_{\ln}^2}$$

The maximum normalized value of the peak current that flows through each power devices is respectively:

$I_{Spkn} = I_{\ln} \cdot (1 + k_{off})$; $I_{Dpkn} = 2 \cdot I_{\ln}$; $I_{SXpkn} = k_{off} \cdot I_{\ln}$; $I_{DXpkn} = k_{off} \cdot I_{\ln}$

5.3.8.2.2 Choosing T0

From the state-plane trajectory, in case of Ilm and Vdcm, can be assumed

$$T_{off} = 2\sqrt{L_x \cdot C_x} \cdot \cos^{-1}\left(1/k_{off}\right) = \frac{T_0}{\pi} \cdot \cos^{-1}\left(1/k_{off}\right)$$

where the choice of Toff is device dependent, and it should be longer than the current fall time of the main switch.

5.3.8.2.3 *Calculation of Lx and Cx*

The values of Lx and Cx can be determined by means of respectively (3) and (4)

5.3.8.2.4 *Timing of auxiliary switch gating signal*

Assuming that the main leg active components are S1 and D2, the switch S1X is gated to achieve both turning-on and turning-off of the main switch S1.

The leading time of gating S1X with respect to S1 can be set as

$$T_{S1Xon} = \frac{T_0}{2} \cdot \left(1 + \frac{k_{off}}{2 \cdot k_{offm}} \cdot I_{ln}\right)$$

where koffm is the value that koff assumes when Vdc=Vdcm and Il=Ilm.

The pulse width of S1X gate on-signal is $PW_{S1Xon} = T_{S1Xon}$ in order to have S1X turn-off and S1 turn-on at the same time.

The leading time of gating S1X with respect to S1 turn-off signal can be chosen as

$$T_{S1Xoff} = T_{S1Xon}$$

whereas the pulse width of S1X gate off-signal is pointed out as

$$PW_{S1Xoff} = T_{S1Xoff}.$$

5.3.8.3 Numerical Example of Resonant Circuit Design for a 100kW PEBB Phase-Leg

Vdcm=800V;

Ilm=191A (pk value at switching frequency, assuming as current ripple 25% of the rated phase current).

ZCT-NZVT
Z0n=0.31, Z0=1.3Ω
Lx=0.725uH; Cx=0.43uF
Toff=800ns, T0=3.5μs

At rated operating condition:
koff=1.225, k=2.225,
Ixpkoff=234A, Ixpk=425A, Vcxpk=1048V,
ISpk=191A IDpk=616A ISXpk=425A IDXpk=191A
Timing:
$PW_{S1X} = T_{S1X} = 2.143us$, $T_{S2X} = 0.875us$, $PW_{S2X} = 3.714us$

At Iln=3/4 (143.25A)
koff=2.3, k=3.3,
Ixpkoff=329.5A, Ixpk=472.7A, Vcxpk=986V,
ISpk=143.25A IDpk=616A ISXpk=472.7A IDXpk=143.25A
Timing:
$PW_{S1X} = T_{S1X} = 2.015us$, $T_{S2X} = 0.875us$, $PW_{S2X} = 4us$

At Iln=1/2 (95.5A)
koff=4.45, k=5.45,
Ixpkoff=425A, Ixpk=529.5A, Vcxpk=924V,
ISpk=95.5A IDpk=616A ISXpk=529.5A IDXpk=95.5A
Timing:
$PW_{S1X} = T_{S1X} = 1.91us$, $T_{S2X} = 0.875us$, $PW_{S2X} = 4.36us$

At Iln=1/4 (47.75A)
koff=10.9, k=11.9,
Ixpkoff=520.5A, Ixpk=568.2A, Vcxpk=862V,
ISpk=47.75A IDpk=616A ISXpk=568.2A IDXpk=47.75A
Timing:

$PW_{S1X} = T_{S1X} = 1.823us$, $T_{S2X} = 0.875us$, $PW_{S2X} = 4.74us$

At Iln=5/4 (238.75A)

koff=(0.58) 1, k=1.58,

Ixpkoff=238.75A Ixpk=377.2A, Vcxpk=1110V,

ISpk=238.75A IDpk=616A ISXpk=377.2A IDXpk=238.75A

Timing:

$PW_{S1X} = T_{S1X} = 2.304us$, $T_{S2X} = 0.875us$, $PW_{S2X} = 3.25us$

IZCT

Z0n=0.715, Z0=3Ω

Lx=1.672uH; Cx=0.186uF

Toff=800ns, T0=3.5μs

At rated operating condition:

koff=1.225, kid=1.4,

Ixpkoff=234A, Vcxpk=1501V,

ISpk=425A IDpk=382A ISXpk=234A IDXpk=234A

Timing:

$PW_{S1Xon} = T_{S1Xon} = PW_{S1Xoff} = T_{S1Xoff} = 2.625us$

At Iln=3/4 (143.25A)

koff=1.733, kid=1.865,

Ixpkoff=248.25A, Vcxpk=1543.5V,

ISpk=391.5A IDpk=286.5A ISXpk=248.25A
IDXpk=248.25A

Timing:

$PW_{S1Xon} = T_{S1Xon} = PW_{S1Xoff} = T_{S1Xoff} = 2.678us$

At Iln=1/2 (95.5A)

koff=2.71, kid=2.8,

Ixpkoff=258.8A, Vcxpk=1575V,

ISpk=354.3A IDpk=191A ISXpk=258.8A IDXpk=258.8A

Timing:

$$PW_{S1Xon} = T_{S1Xon} = PW_{S1Xoff} = T_{S1Xoff} = 2.718us$$

At Iln=1/4 (47.75A)

koff=5.55, kid=5.6,

Ixpkoff=265A, Vcxpk=1593.6V,

ISpk=312.7A IDpk=85A ISXpk=265A IDXpk=265A

Timing:

$$PW_{S1Xon} = T_{S1Xon} = PW_{S1Xoff} = T_{S1Xoff} = 2.741us$$

At Iln=5/4 (238.75A)

koff=(0.906) 1 kid=1.119,

Ixpkoff=238.75 Vcxpk=1515V,

ISpk=404.7A IDpk=477.5 ISXpk=238.75 IDXpk=238.75

Timing:

$$PW_{S1Xon} = T_{S1Xon} = PW_{S1Xoff} = T_{S1Xoff} = 2.643us$$

Table 5-6 Theoretical calculation results and simulation results (between parenthesis)

	ZCT-NZVT				IZCT			
Main current [A]	Main switch current [A]	Main diode current [A]	Auxiliary switch current [A]	Auxiliary diode current [A]	Main switch current [A]	Main diode current [A]	Auxiliary switch current [A]	Auxiliary diode current [A]
191	191 (191)	616 (621)	425 (430)	191 (191)	425 (414)	382 (360)	234 (223)	234 (197)
143.25	143.25 (143.25)	616 (602)	472.7 (458)	143.25 (143.25)	391.5 (370)	286.5 (270)	248.25 (227)	248.25 (200)
95.5	95.5 (95.5)	616 (582)	529.5 (486)	95.5 (95.5)	354.3 (329)	191 (179)	258.8 (233)	258.8 (206)
47.25	47.25 (47.25)	616 (564)	568.2 (516)	47.25 (47.25)	312.7 (292)	95.5 (89)	265 (244)	265 (216)
238.75	238.75 (238.75)	616 (642)	377.2 (403)	238.75 (238.25)	477.5 (442)	477.5 (450)	238.75 (211)	238.75 (238.75)

5.3.8.4 Numerical and Simulation Results (Pspice PEBB phase leg models)

5.3.8.4.1 ZCT-NZVT (rated conditions of operation)

Resonant inductor current and resonant capacitor voltage

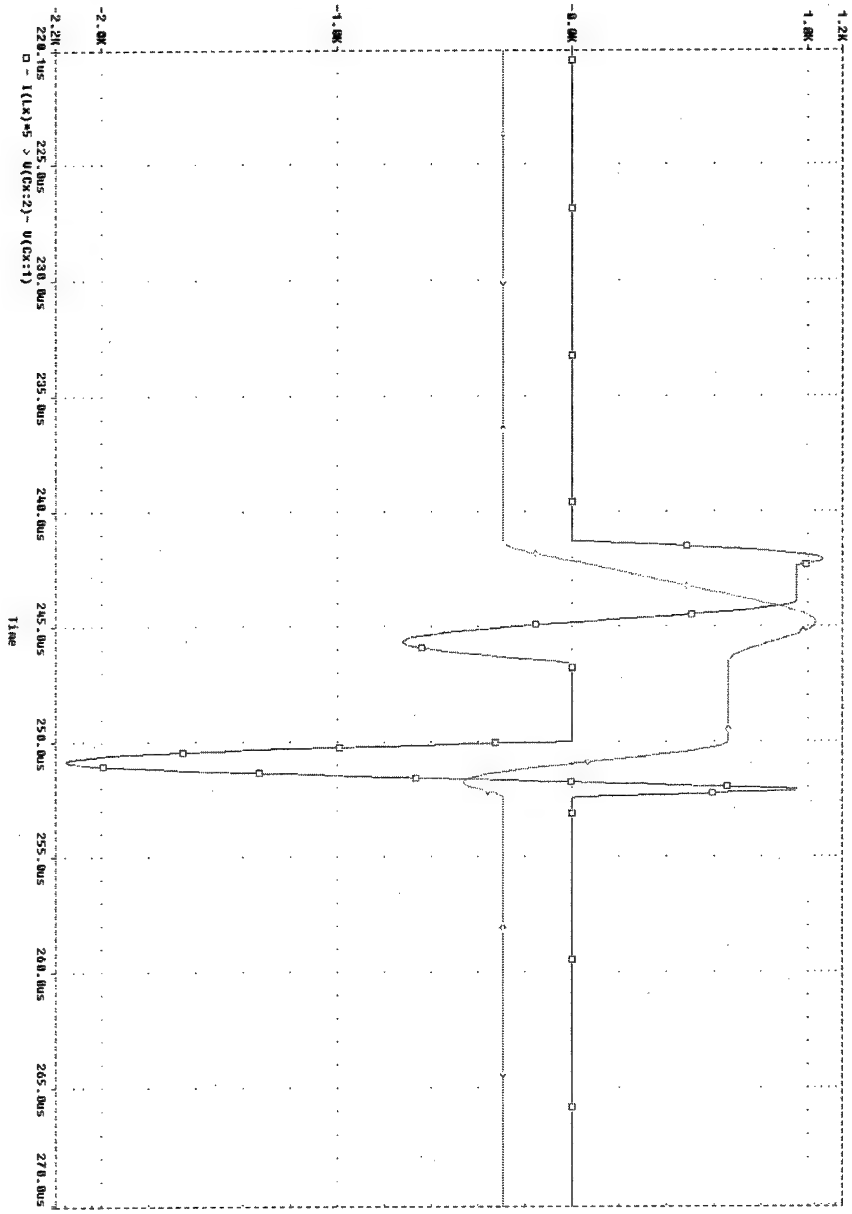


Fig. 5-38 Main leg currents

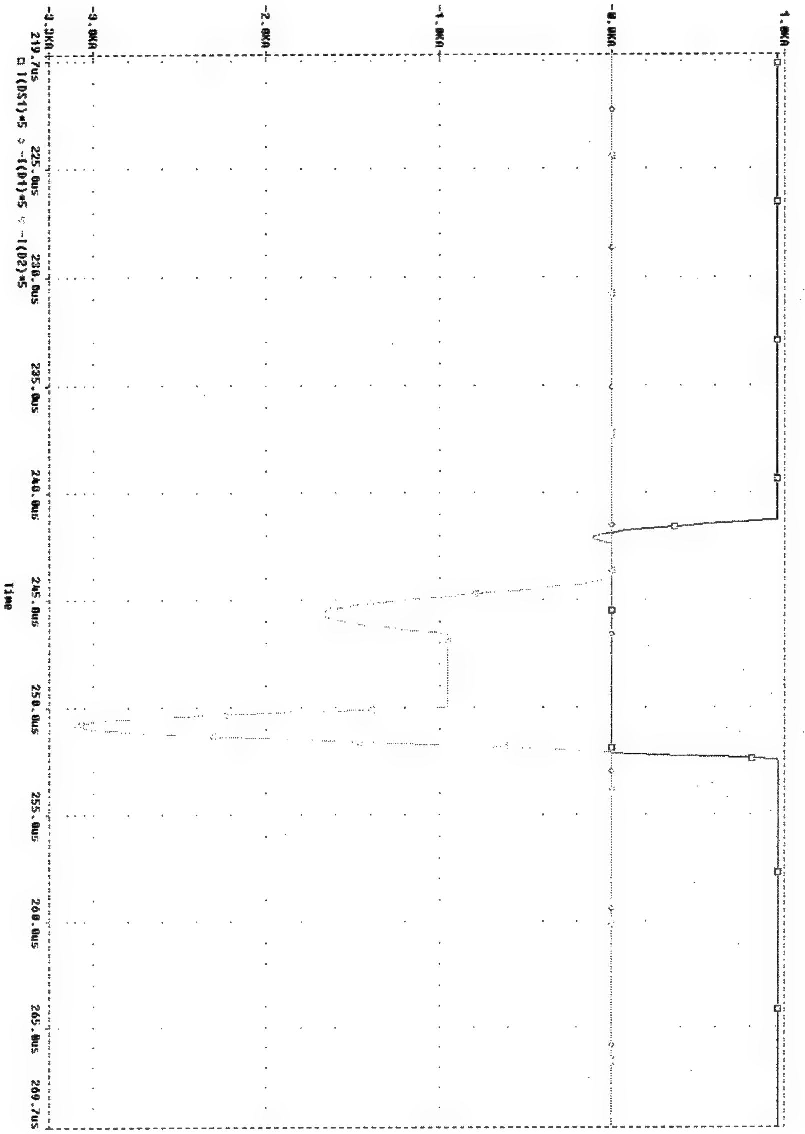


Fig. 5-39 Auxiliary leg currents

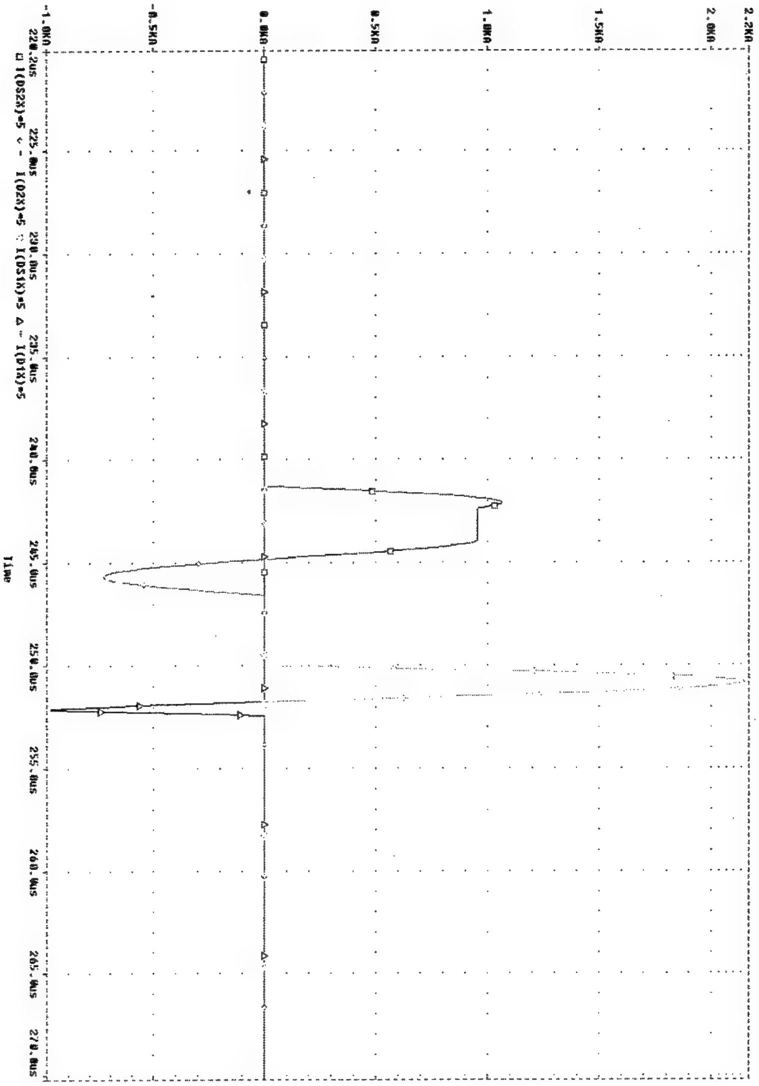


Fig. 5-40 IZCT (rated conditions of operation)

5.3.8.4.2 *Resonant inductor current and resonant capacitor voltage*

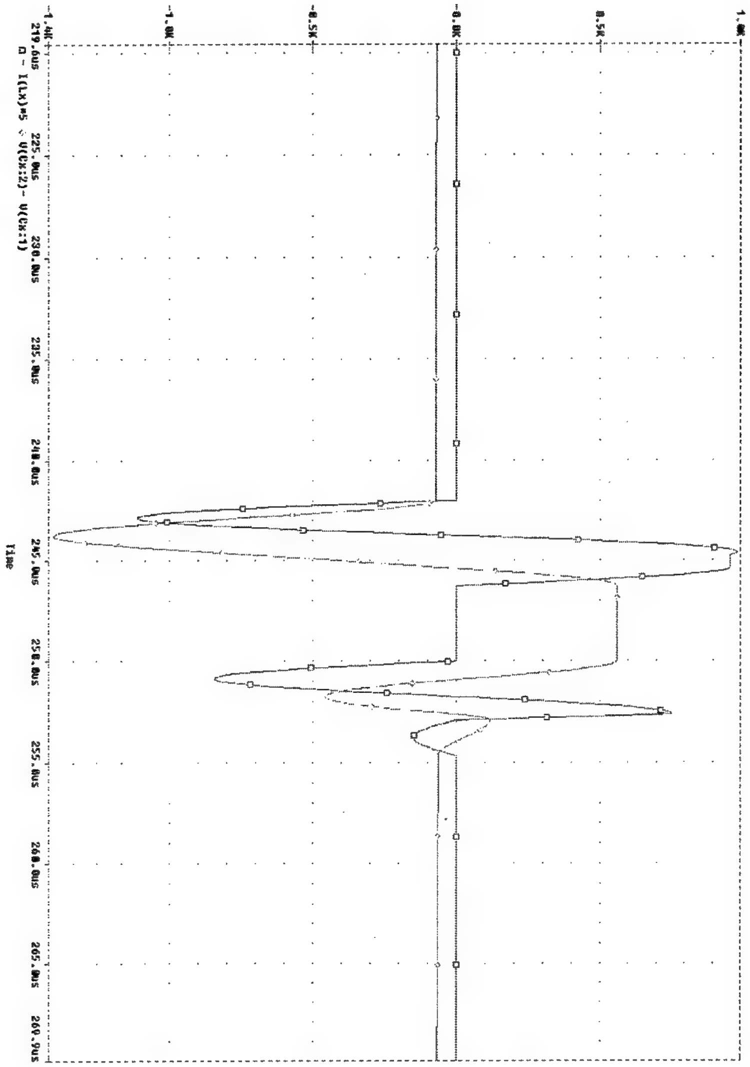


Fig. 5-41 Main leg currents

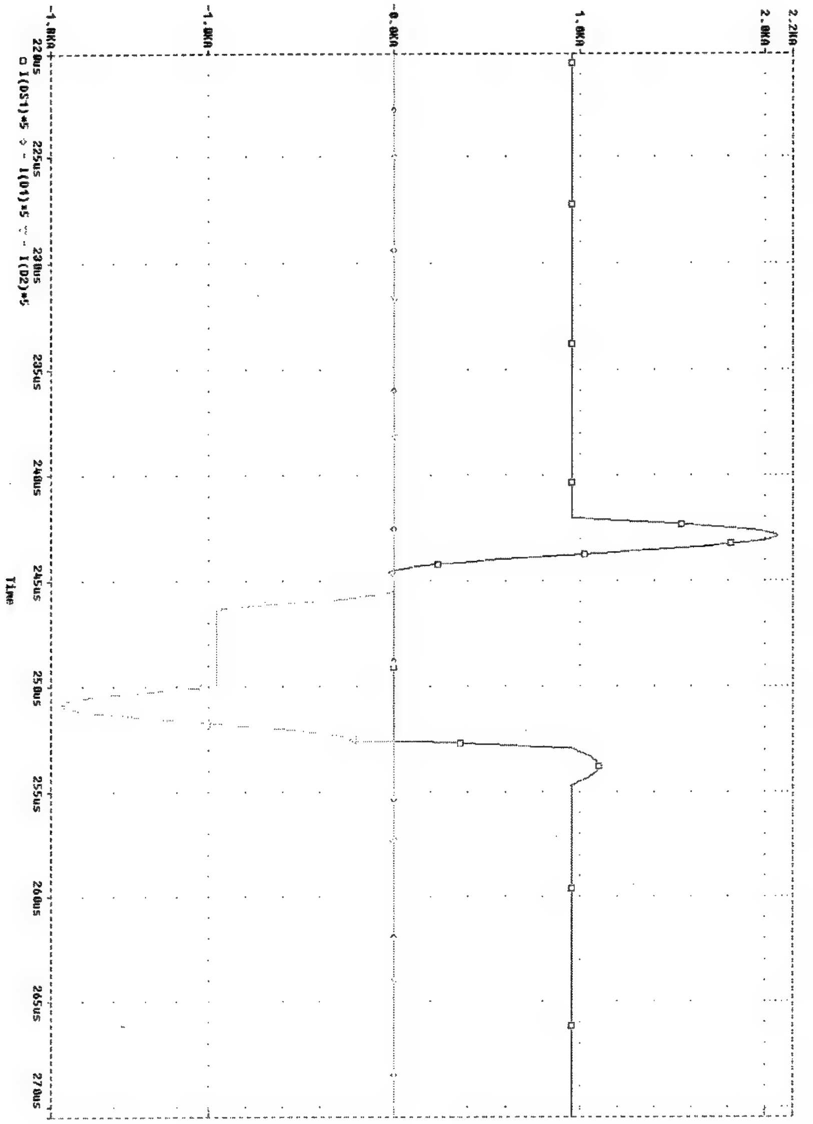


Fig. 5-42 Auxiliary leg currents

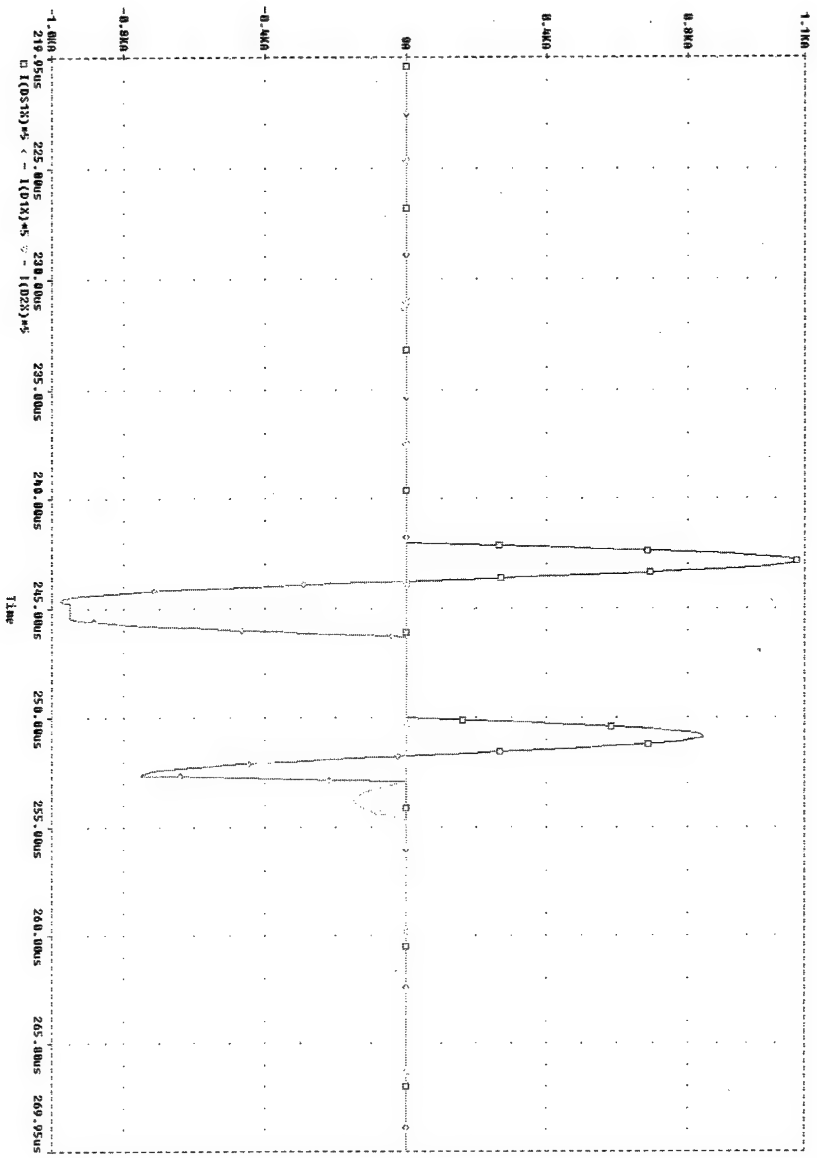


Fig. 5-43 ZCT-NZVT (rated conditions of operation)

5.3.8.5 Comparison Between the ZCT-NZVT and the IZCT Techniques

On the basis of the normalized approach, IZCT and ZCT-NZVT resonant circuit designs can be compared.

Resonant capacitance value:

lower in IZCT topology than in ZCT-NZVT one, thus the total resonant capacitance can be mainly affected by parasitic elements in IZCT scheme.

Resonant capacitor peak voltage level:

close to 2 times the dc link voltage for IZCT technique, 1.3-1.4 times the dc link voltage for ZCT-NZVT technique.

Resonant inductance value:

lower in ZCT-NZVT topology than in IZCT one, thus the total resonant inductance can be mainly affected by parasitic elements in ZCT-NZVT scheme.

Resonant inductor peak current level:

2.3-2.4 times the maximum current subjected to ZCT turn-off for ZCT-NZVT technique, 1.3-1.4 times the maximum current subjected to ZCT for IZCT topology.

Main switch peak current:

2.3-2.4 times the maximum current subjected to ZCT turn-off for IZCT scheme, equal to the maximum current subjected to ZCT for ZCT-NZVT configuration.

Main diode peak current:

3.3-3.4 times the maximum current subjected to ZCT turn-off for ZCT-NZVT technique, 2 times the maximum current subjected to ZCT for IZCT topology.

Auxiliary switch peak current:

2.3-2.4 times the maximum current subjected to ZCT turn-off for ZCT-NZVT technique, 1.3-1.4 times the maximum current subjected to ZCT for IZCT topology

Auxiliary diode peak current:

1.3-1.4 times the maximum current subjected to ZCT turn-off for IZCT scheme, equal to the maximum current subjected to ZCT for ZCT-NZVT configuration.

Timing of the auxiliary switch gate signals:

very easy for the IZCT technique, more complicated in ZCT-NZVT technique.

Behavior at main current lower than the maximum current subjected to ZCT turn-off:

turn-off hard switching of the auxiliary switch in ZCT-NZVT scheme at 25% of the maximum current subjected to ZCT turn-off, regular even at 25% of the maximum current subjected to ZCT turn-off for IZCT configuration.

Behavior at main current higher than the maximum current subjected to ZCT turn-off:

partial turn-off hard switching of the main switch in ZCT-NZVT scheme at 125% of the maximum current subjected to ZCT turn-off, partial turn-off hard switching of the main switch in IZCT scheme at 125% of the maximum current subjected to ZCT turn-off (however, the hard switched current is lower than in ZCT-NZVT case).

5.3.8.6 Design of the Resonant Circuit

5.3.8.6.1 Resonant Inductor Design

Design input parameters:

inductance value Lx;

rated peak current (Ixpk) and rated rms current (Ixrms);

operating frequency (f0);

In order to carry on the design of the resonant inductor the following expressions are considered:

$$V_e = A_e \cdot l_e = \frac{\mu_0 \mu \cdot L_x \cdot I_{xpk}^2}{B_{pk}^2}$$

$$N = \frac{L_x \cdot I_{xpk}}{A_e \cdot B_{pk}}$$

$$L_x = \frac{\mu_0 \mu \cdot A_e \cdot N^2}{l_e}$$

$$A_w = \frac{L_x \cdot I_{xpk} \cdot I_{xrms}}{k_{cu} \cdot J_{xrms} \cdot A_e \cdot B_{pk}}$$

where Ve, Ae, le and μ0·μ are respectively the volume, the cross section, the path length and the magnetic permeability of the material in which the magnetic energy is stored; Bpk is the peak flux density in the core; Aw, N, kcu and Jxrms are respectively the required winding area, the number of turns, the fill factor and the current density considered for the inductor winding; from simulation results Ixrms is equal to 76.7A and 53.3A respectively for ZCT-NZVT and IZCT soft-switching techniques.

The core material is chosen on the basis of the operating frequency; thus, the value of Bpk and the magnetic permeability are found from data-sheet of the selected material.

The core size closest to the previous result is chosen among the options available on the market; then, by using the previous expressions the actual values of Bpk, N, Lx and Aw are known.

Molypermalloy powder (MPP) core, high flux powder (HFP) core, Metglas core and ferrite core have been investigated for the resonant inductor of the proposed soft-switching techniques; in each design the copper fill factor and the winding current density have been considered equal respectively to 0.6 and 6A/mm2.

MPP Core Inductor (Magnetics Inc.) [5]

5.3.8.6.1.1 ZCT-NZVT technique

Physical Characteristics						Results from (20) through (23)			
Part Num.	N. of Pcs.	Ext. Dim. (wxhxl mm3)	μ	Ae (cm2)	le (cm)	Bpk (T)	Lx (μH)	N	Aw (cm2)
55190-A2	3	57.2x57.2x45.6	14	6.87	12.5	0.179	0.87	3	0.64
55902-A2	2	77.8x77.8x31.8	14	4.54	19.95	0.15	0.64	4	0.85
55441-A2	4	46.7x46.7x72	14	7.96	10.74	0.139	0.521	2	0.43

From material data-sheet the core losses can be roughly calculated:

expected core losses: 120mW (for all the three options)

5.3.8.6.1.2

5.3.8.6.1.3 IZCT technique

Physical Characteristics						Results from (20) through (23)			
Part Num.	N. of Pcs.	Ext. Dim. (wxhxl mm3)	µ	Ae (cm2)	le (cm)	Bpk (T)	Lx (µH)	N	Aw (cm2)
55869-A2	2	77.8x77.8x25.4	14	3.54	20.0	0.145	1.525	7	1.02
55190-A2	2	57.2x57.2x30.4	14	4.58	12.5	0.165	1.611	5	0.75
55441-A2	3	46.7x46.7x54	14	5.97	10.74	0.153	1.564	4	0.59

Expected core losses: 100mW (for all the three options)

HFP Core Inductor (Magnetics Inc.)

5.3.8.6.1.4 ZCT-NZVT technique

Physical Characteristics						Results from (20) through (23)			
Part Num.	N. of Pcs.	Ext. Dim. (wxhxl mm3)	µ	Ae (cm2)	le (cm)	Bpk (T)	Lx (µH)	N	Aw (cm2)
58933-A2	2	26.9x26.9x22.4	14	1.308	6.35	0.47	0.58	4	0.85
58257-A2	1	39.9x39.9x14.5	14	1.072	9.84	0.46	0.69	6	1.28

Expected core losses: 100W (for all the two options)

5.3.8.6.1.5 IZCT technique

Physical Characteristics						Results from (20) through (23)			
Part Num.	N of Pcs.	Ext. Dim. (wxhxl mm3)	μ	Ae (cm2)	le (cm)	Bpk (T)	Lx (μH)	N	Aw (cm2)
58327-A2	1	35.8x35.8x10.5	14	0.678	8.98	0.504	1.606	11	1.63
58256-A2	1	39.9x39.9x14.5	26	1.072	9.84	0.545	1.743	7	1.03

Expected core losses: 100W (for the first core); 175W (for the second core)

Metglas Core Inductor (Magnetics Inc.)

5.3.8.6.1.6 ZCT-NZVT technique

Physical Characteristics					Results from (20) through (23)				
Part Num.	N. of Pcs.	Ext. Dim. (wxhxl mm3)	μ	Ae (cm2)	le (cm)	Bpk (T)	Lx (μH)	N	Aw (cm2)
MC0012	1	34.95x50.8x12.7	1	1.06	0.17	0.95	0.725	3	0.64

Expected core losses: 72.5W

5.3.8.6.1.7 IZCT technique

Physical Characteristics					Results from (20) through (23)				
Part Num.	N. of Couples	Ext. Dim. (wxhxl mm3)	μ	Ae (cm2)	le (cm)	Bpk (T)	Lx (μH)	N	Aw (cm2)
MC0009	1	28.58x49.3x12.7	1	0.907	0.17	0.864	1.672	5	0.75

Expected core losses: 47.5W

Ferrite Core Inductor (Magnetics Inc.) [9]

5.3.8.6.1.8 ZCT-NZVT technique

Physical Characteristics					Results from (20) through (23)				
Part Num.	N. of Couples	Ext. Dim. (wxhxl mm3)	μ	Ae (cm2)	le (cm)	Bpk (T)	Lx (μH)	N	Aw (cm2)
P45528-EC	2	54.9x55.2x41.2	1	7.0	0.48	0.221	0.725	2	0.43
P44924-EC	2	49.1x47.6x31.3	1	5.14	0.36	0.3	0.725	2	0.43
P45530-EC	1	54.9x55.2x24.6	1	4.17	0.65	0.246	0.725	3	0.64

Expected core losses: 15W (for the first option); 10W (for both the second and the third core)

5.3.8.6.1.9

5.3.8.6.1.10

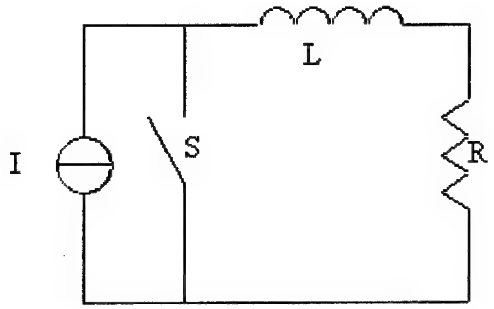
5.3.8.6.1.11 IZCT technique

Physical Characteristics					Results from (20) through (23)				
Part Num.	N. of Couples	Ext. Dim. (wxhxl mm3)	μ	Ae (cm2)	le (cm)	Bpk (T)	Lx (μH)	N	Aw (cm2)
P45530-EC	1	54.9x55.2x24.6	1	4.17	0.5	0.235	1.672	4	0.85

Expected core losses: 10W

5.3.8.6.2 A simple circuit for inductor experimental testing

The following circuital configuration can be adopted to test the inductance value of the assembled inductors.



A dc current generator supplies the inductor L under testing in series with a resistor R having resistance value very high if compared to the inductor resistance. At time t=0 the switch S is closed, by monitoring the current flowing through the resistor and the voltage across the resistor it is possible to achieve the indirect measurement of the inductance. In fact,

$$L = \frac{\phi}{I}$$

where

$$\phi = \int V_R \cdot dt$$

can be easily known by using modern scopes.

5.3.8.7 Resonant Capacitor Selection (Electronic Concepts Inc.)

5.3.8.7.1 ZCT-NZVT Technique

5PT46L104	0.1µF, 1200V	4 Pcs.	or
PT88BN224	0.22µF, 1200V	2 Pcs.	or
PT88BN394	0.39µF, 1200V	1 Pc.	

5.3.8.7.2 IZCT Technique

5PT46M104 + 5PT46M823	0.1µF+0.082µF, 1500V	1+1 Pcs.
MT88BT184	0.18µF, 1600V	1 Pc.

5.3.9 Experimental Verification of Soft Switched PEBBs

The old phase-legs developed in the previous project were upgraded to accommodate for soft switching commutation. The resultant phase leg structure is shown in Fig. 5-44, clearly depicting both main and auxiliary switches, as well as the L-C series resonant tank. Fig. 5-45 shows actual pictures taken from one of the upgraded PEBBs, clearly showing its main components, including the current sensor. Fig. 5-46 shows the experimental setup used to test both studied algorithms, and Fig. 5-47 and Fig. 5-48 show experimental results obtained. Particularly Fig. 5-47 shows the on off transitions for the I-ZCT techniques and Fig. 5-48 the on off transitions for the ZV-ZCT technique. The main result attained was the reduction of the commutation losses, which naturally increases the efficiency of the PEBB and allows for high frequency operation without increasing the thermal requirements of the converter system.

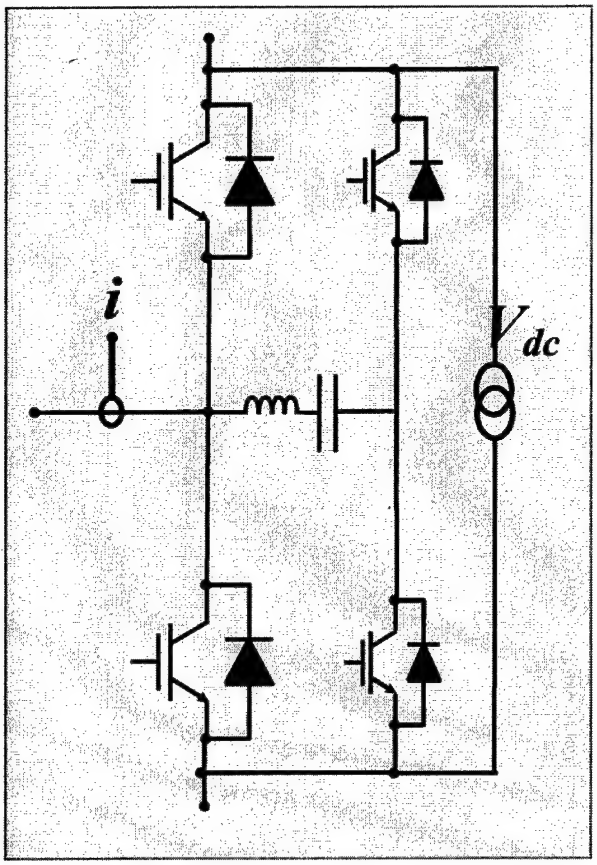


Fig. 5-44 Soft switching PEBB schematic built from old phase-leg by adding auxiliary resonant circuit.

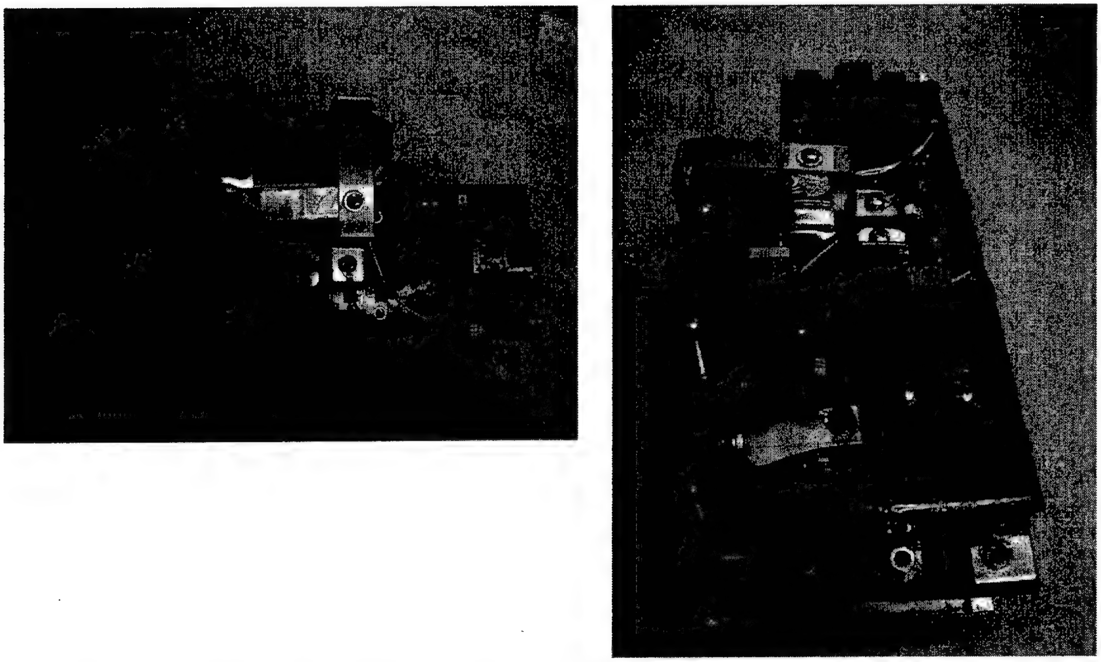


Fig. 5-45 Downwards and lateral views of Softswitching PEBB.

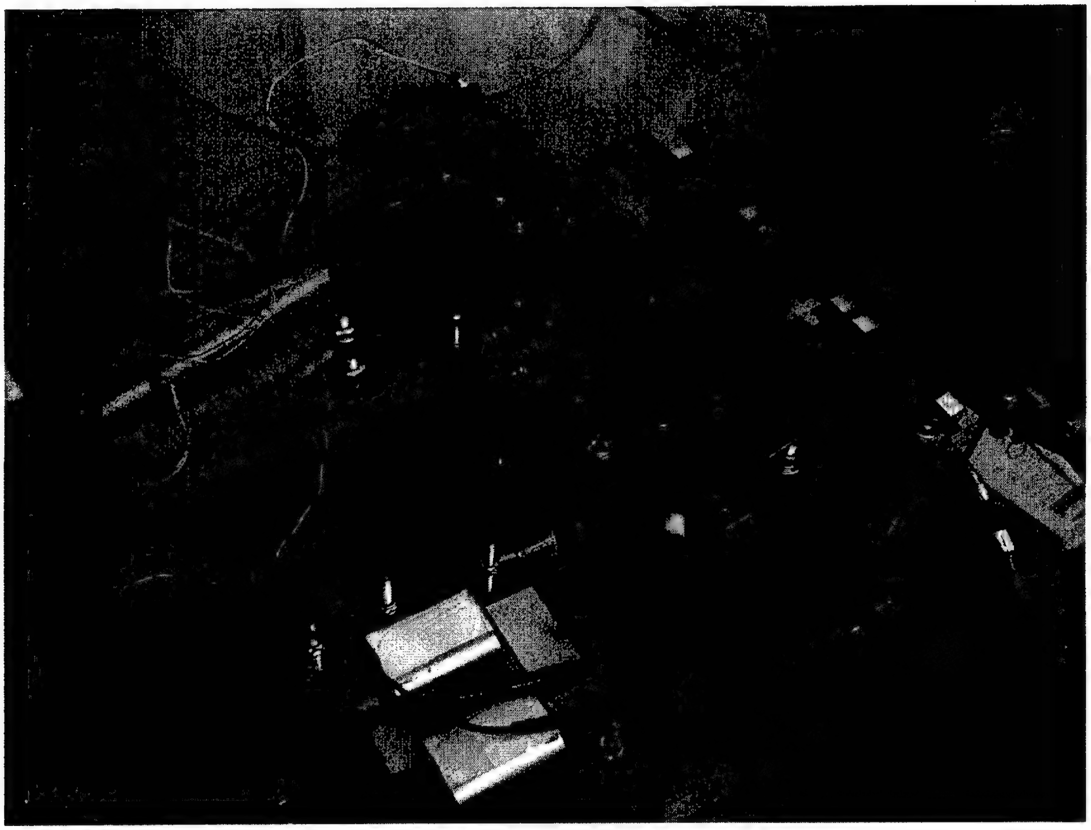


Fig. 5-46 Experimental setup used to test appropriate operation of the softswitching PEBBs.

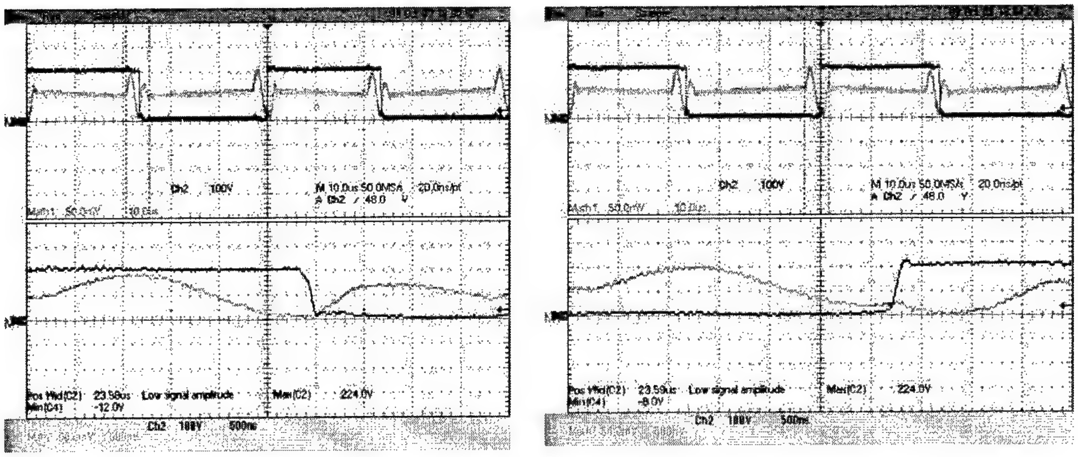


Fig. 5-47 Turn on and turn off transients for IZCT softswitching algorithm.

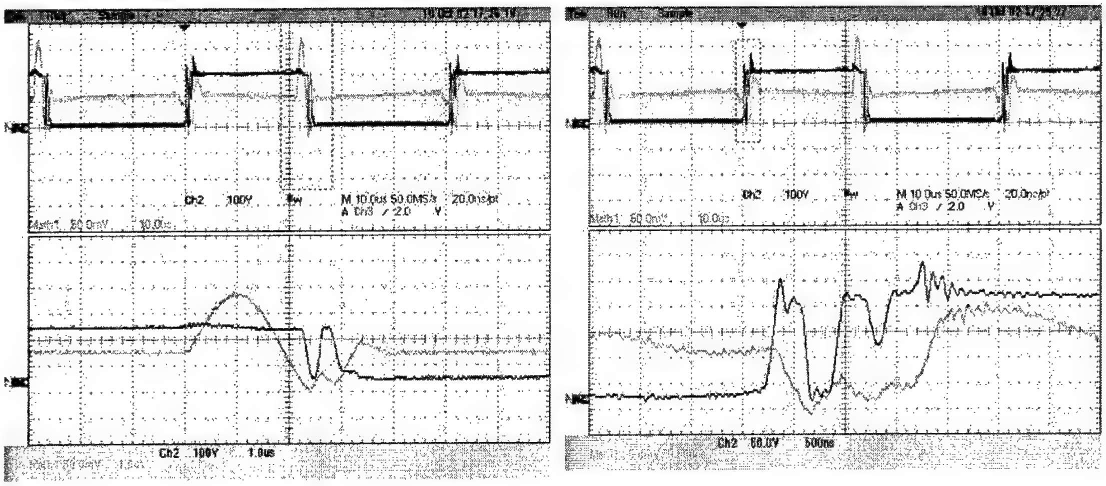


Fig. 5-48 Turn on and turn off transients for ZV-ZCT softswitching algorithm.

shows the cabinet we used to run the test. Fig. 5-50 illustrates how a soft-switched PEBB can be plugged into the cabinet. The circuit diagram of the DC-DC test is shown in Fig. 5-51. All the tests run at:

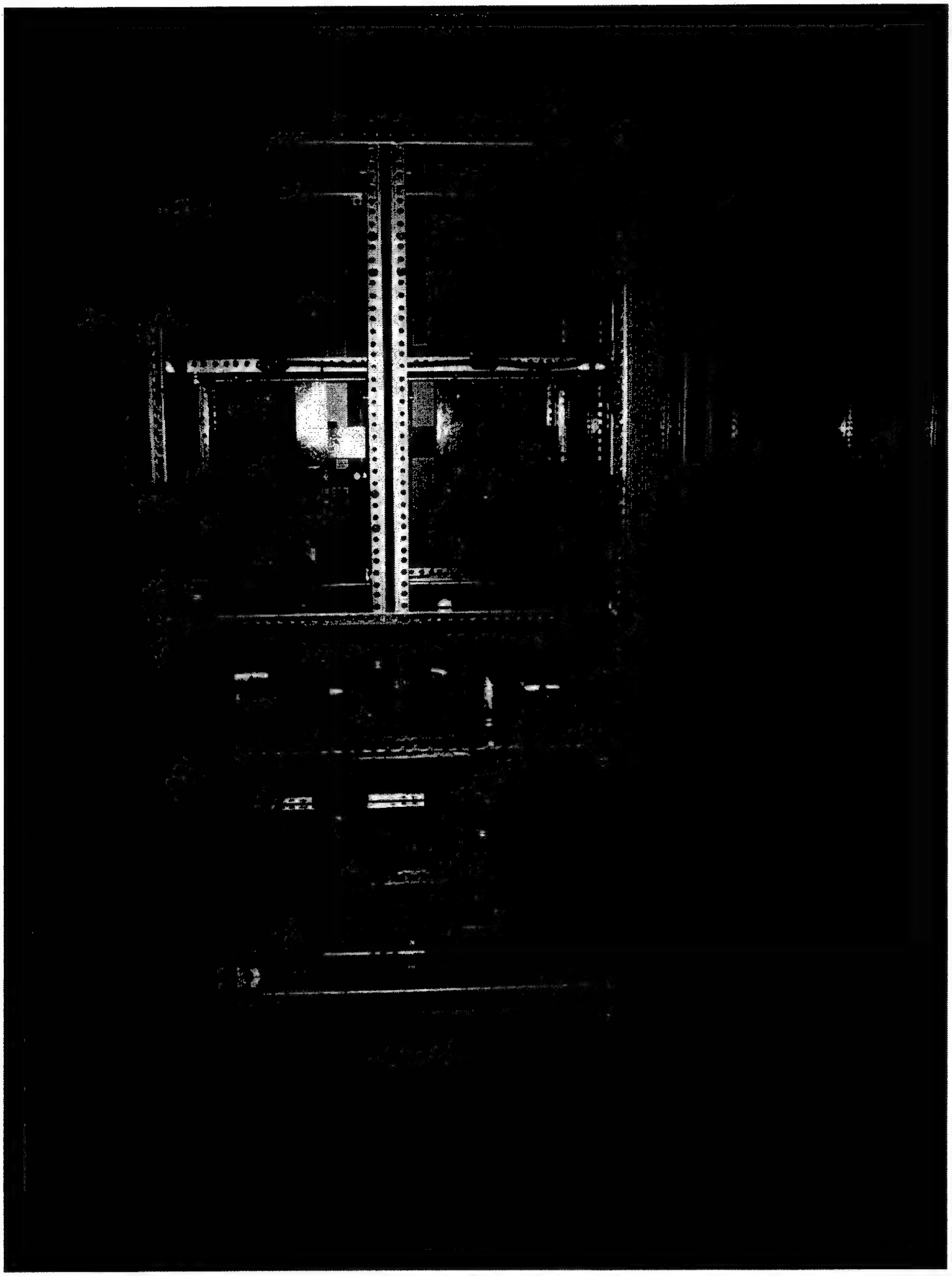
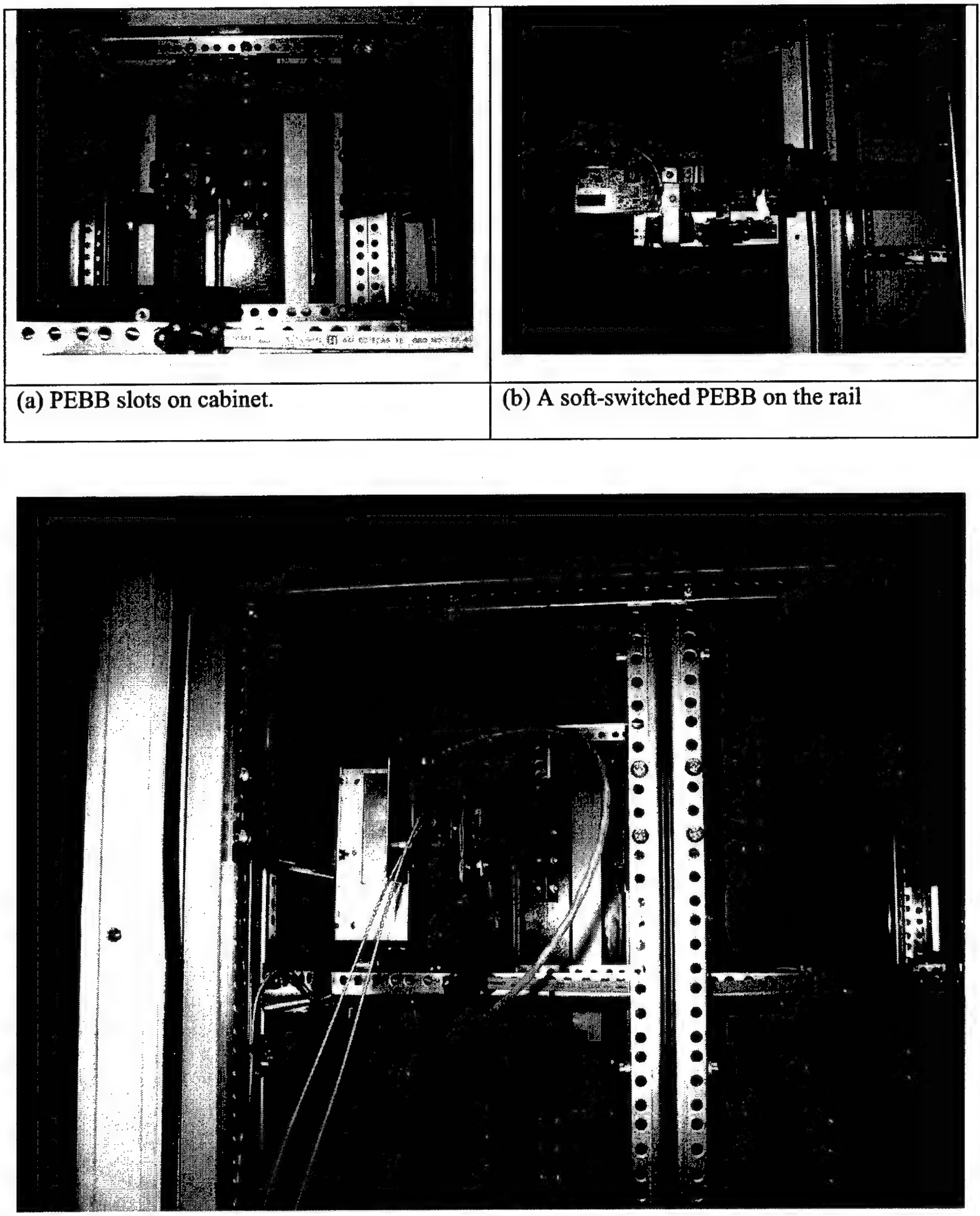


Fig. 5-49 PnP compliant cabinet.



(a) PEBB slots on cabinet.

(b) A soft-switched PEBB on the rail

(c) A soft-switched PEBB plugged into the cabinet.

Fig. 5-50 Test soft-switched PEBB on the cabinet.

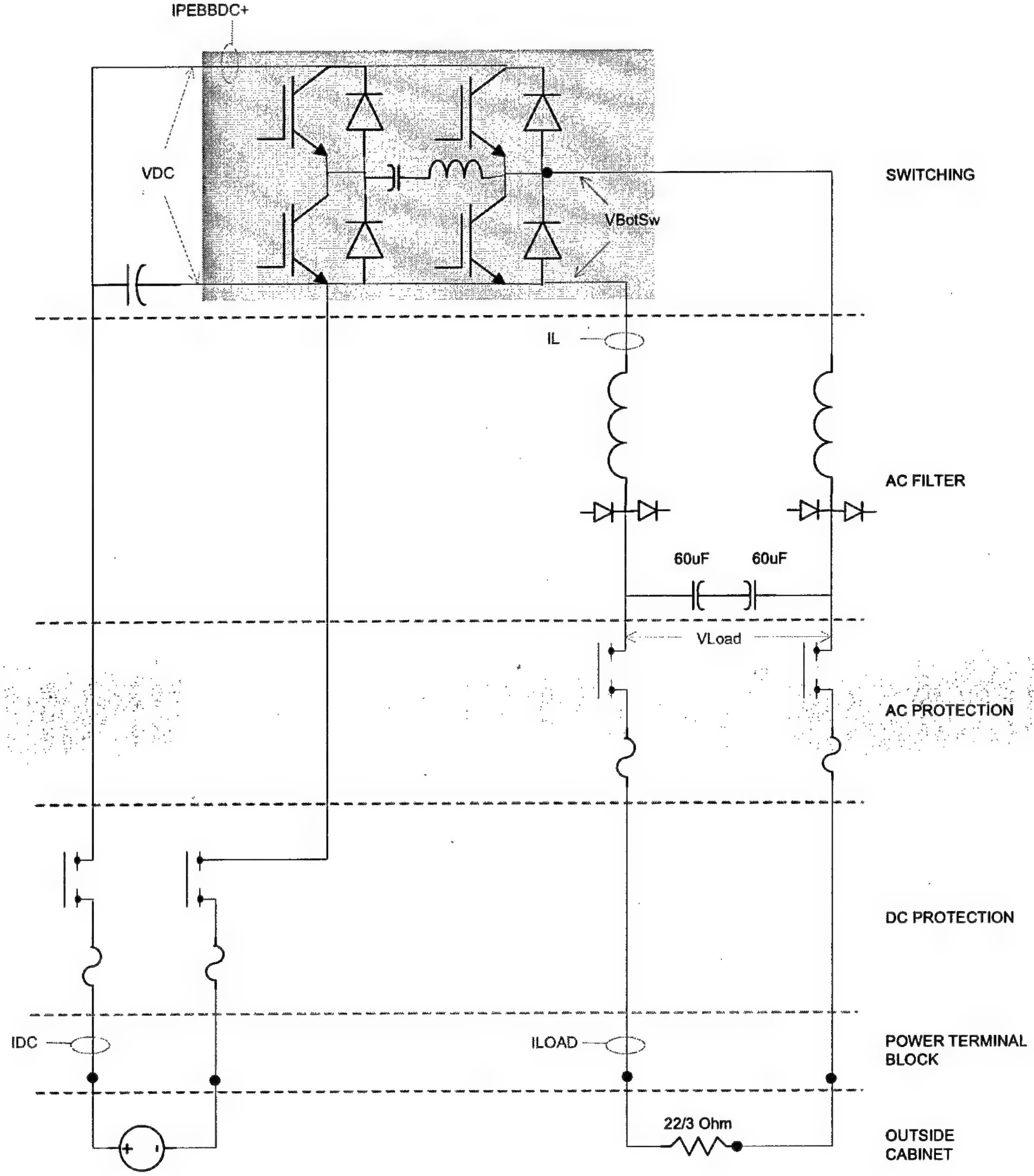


Fig. 5-51 Circuit diagram of DC-DC test for soft-switched PEBB.

We run the DC-DC test by generating constant duty cycle from the integrated hardware on the soft-switched PEBB to verify whether the PEBB is able to work with cabinet properly. Fig.

5-52 shows the experimental waveform of this test. C1 is the input voltage; C2 is the voltage across the bottom main switch; M1 is the output inductor current.

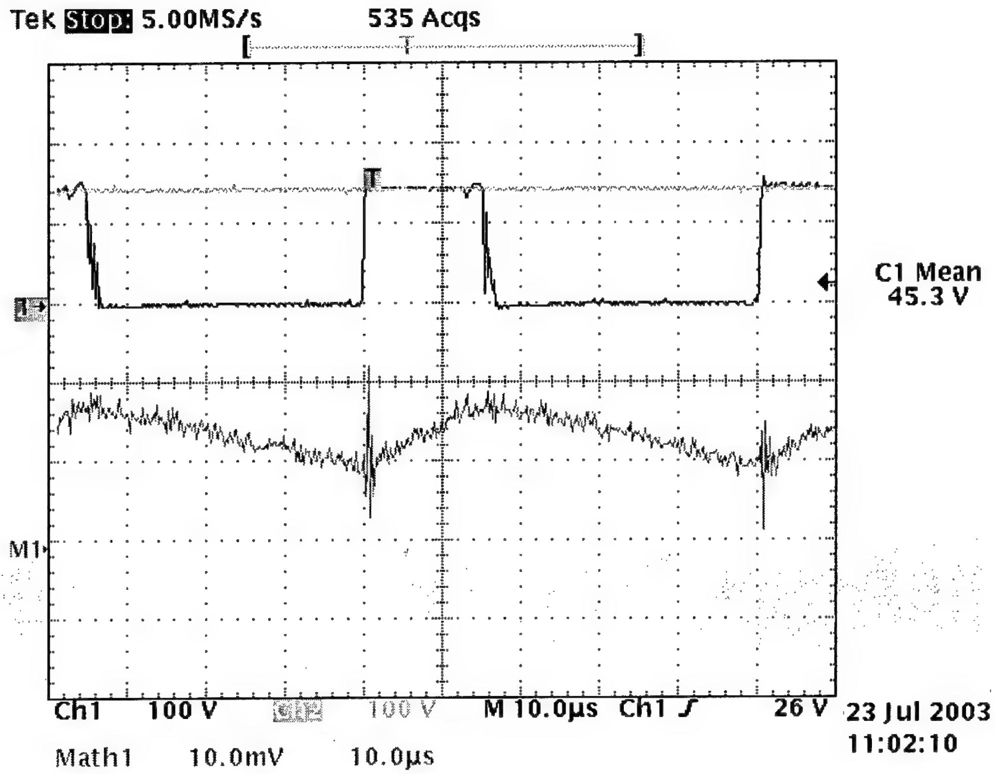


Fig. 5-52 DC-DC test waveforms of soft-switched PEBB, with hardware manager generating constant duty cycle.

We also did a similar DC-DC test with the PWM control information sent from the UC to the hardware manager through PESNet. Fig. 6 6 shows the experimental waveforms with the constant duty cycle (=0.4) received from the UC; while Fig. 6 7 shows the waveforms resulted from the duty cycle change to 0.8 at the UC.

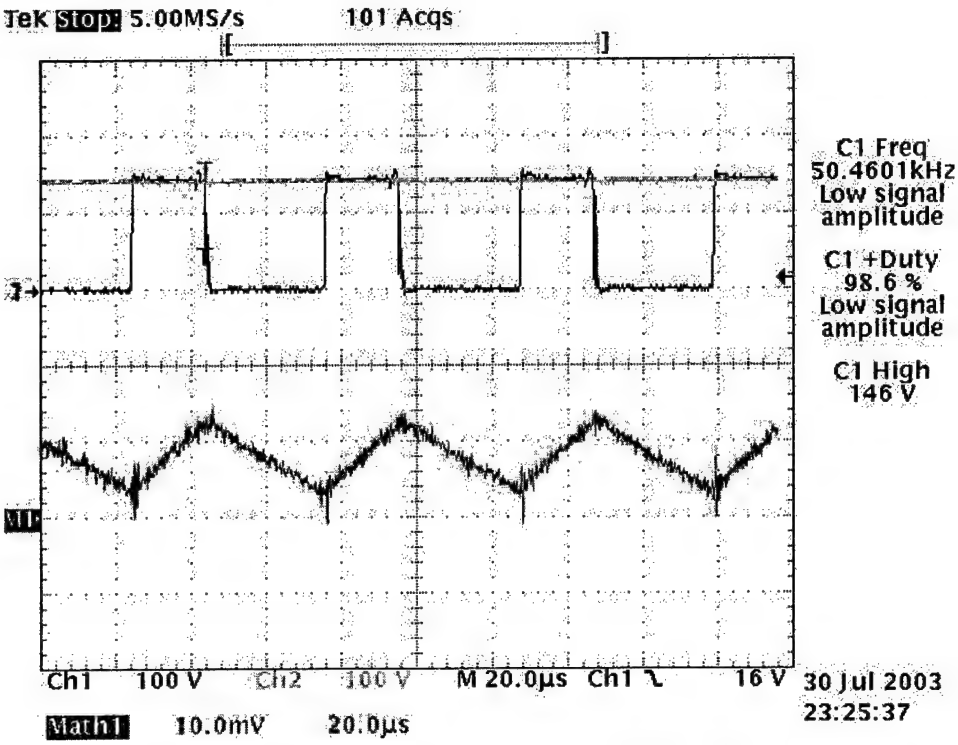


Fig. 5-53 DC-DC test waveforms of soft-switched PEBB, with a constant duty cycle (0.4) sent from UC.

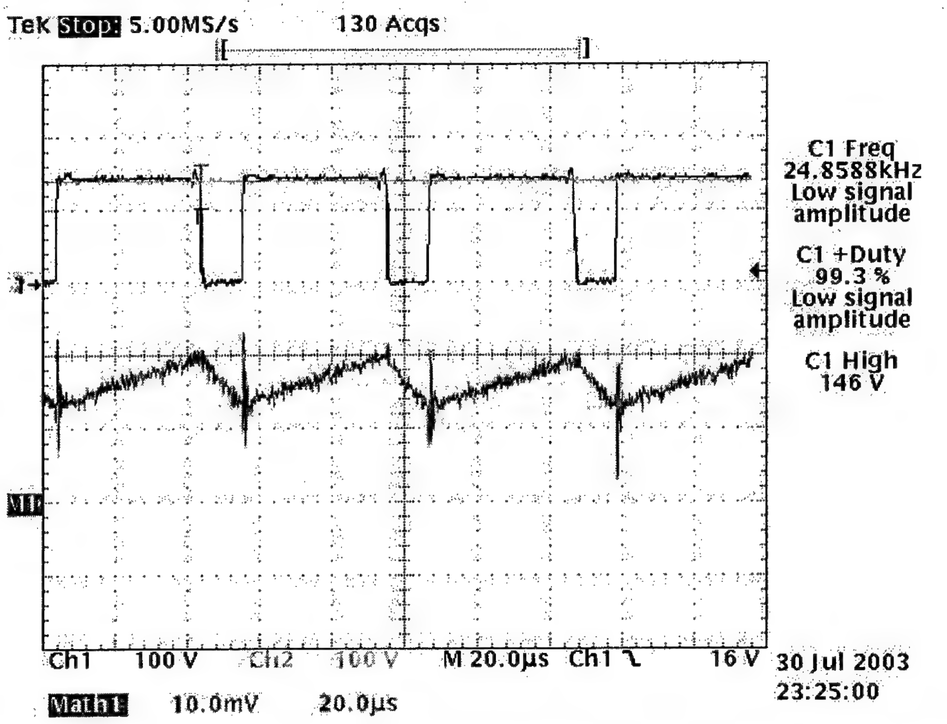


Fig. 5-54 DC-DC test waveforms of soft-switched PEBB, with a constant duty cycle (0.8) sent from UC.

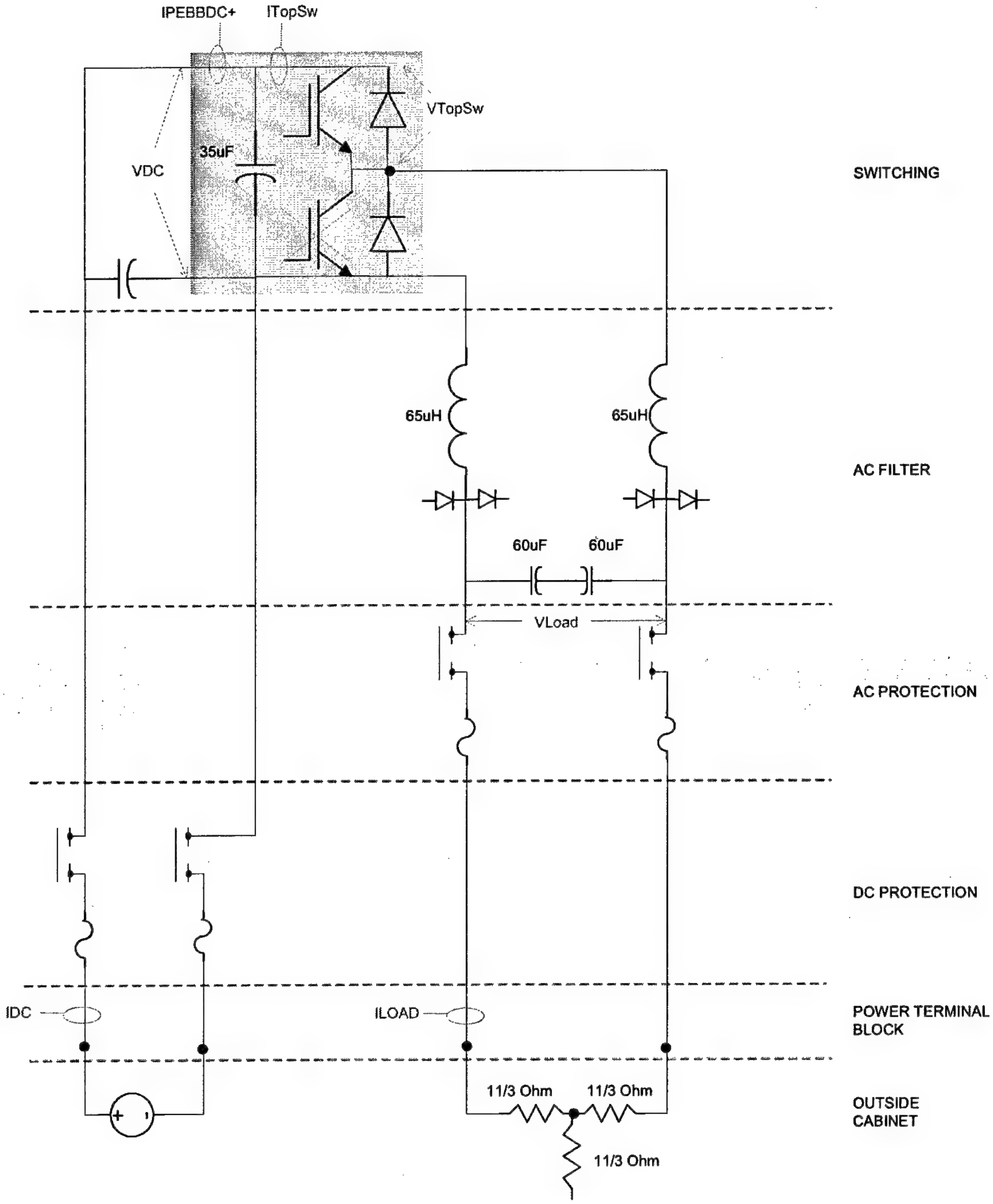


Fig. 5-55 Experimental set-up schematic for testing the new PEBB modules

5.4 Experimental Validation of New PEBB Modules

In order to verify the proper operation of power stage and new PEBB modules, tests were run up to a DC link voltage of 200V. Particularly, the new PEBB was connected to operate in a buck DC/DC converter mode with the bottom switch disabled all the time, and the top switch switching at 20 kHz.

5.4.1 Buck dc-dc operation

The configuration used for the test is the same as that shown in Fig. 5-55. The variables measured were Vdc, VTopSw, Idc, and Vload. The current was set at 5 A/10mV and was measured using a Tektronix Hall-Effect gun. All voltage measurements were made using Tektronix high voltage differential probes.

Channel	Description	Scale	Notes
1	VTopSwitch	200V/div	
2	VDC	200V/div	
3	Idc	5A/div	(1div = 10mV), Choke on BNC cable
4	Vload	100V/div	

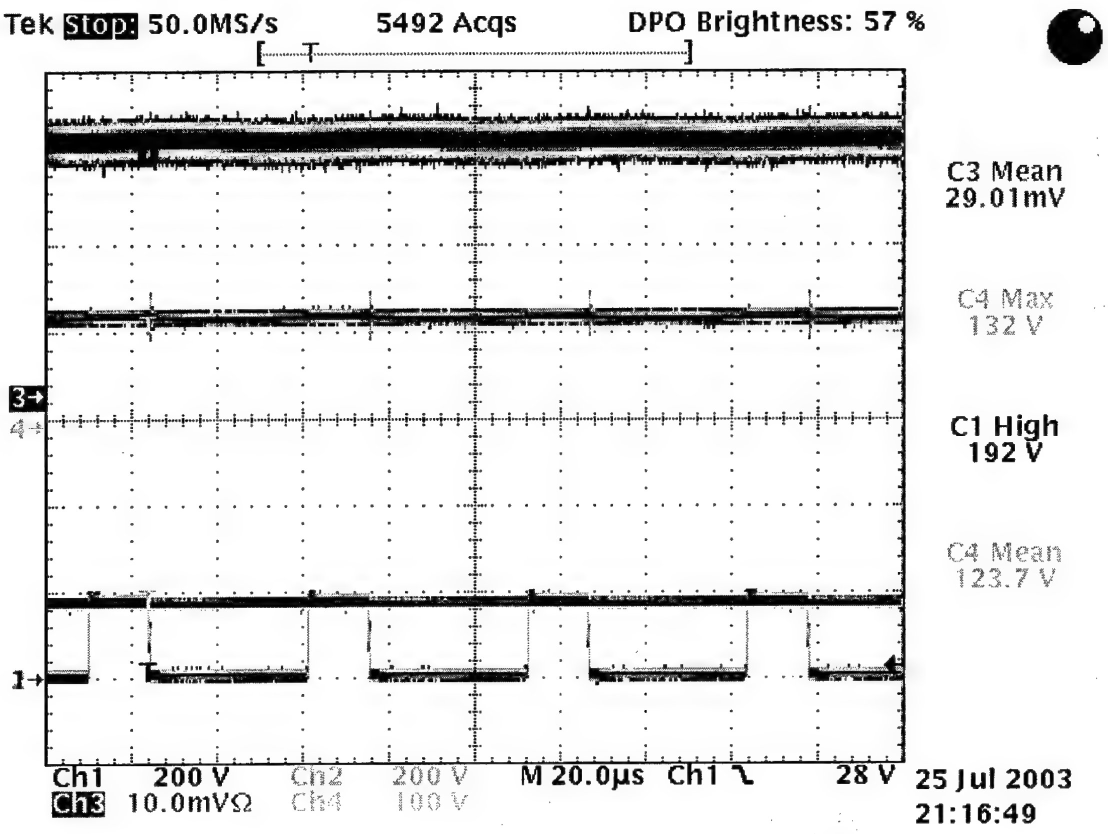


Fig. 5-56 New PEBB operating in Buck dc-dc converter mode.

An expanded view of the switch turn off is shown below.

Channel	Description	Scale	Notes
1	VTopSwitch	200V/div	
2	VDC	200V/div	
3	Idc	5A/div	(1div = 10mV), Choke on BNC cable
4	Vload	100V/div	

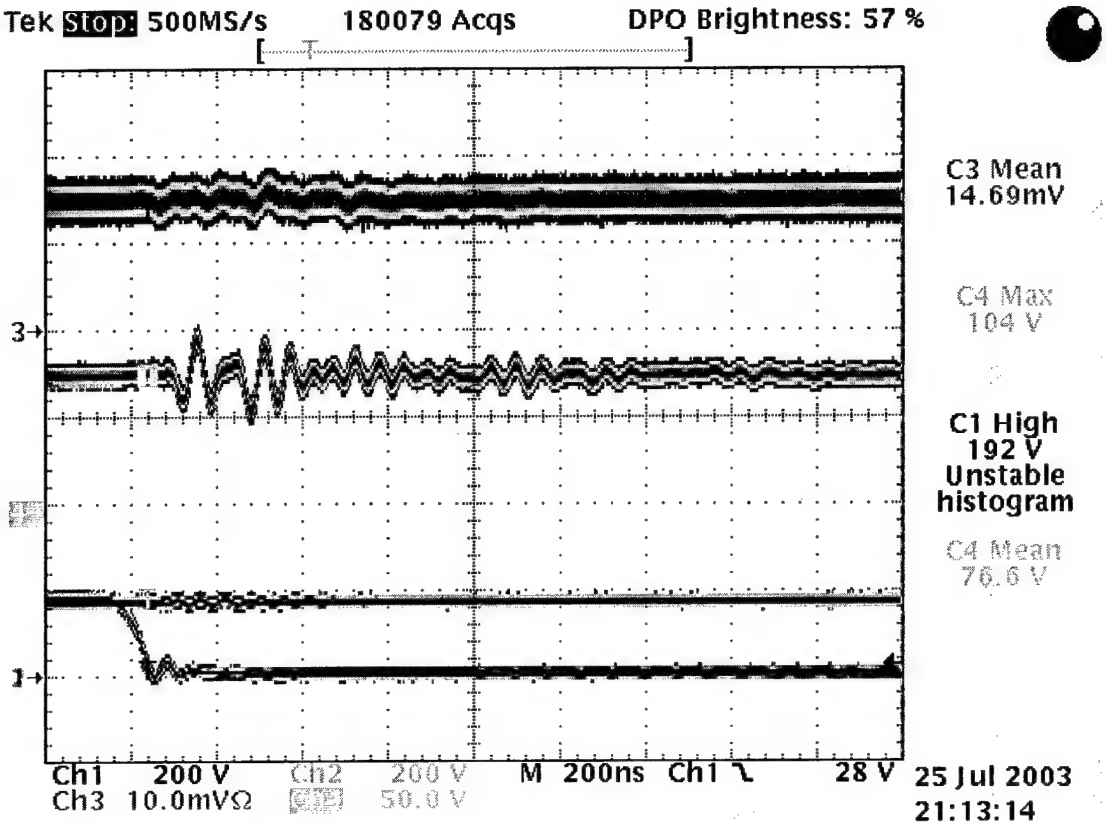


Fig. 5-57 Turn off transient of new PEBB operating in Buck dc-dc converter mode.

5.4.2 Pulse Test Results

A two-pulse test was performed in order to show the phase leg could safely switch at high power and EMI levels. The output of Phase C was shorted to the negative DC rail, making the load of Phase C to be the 216uH inductor. The schematic shown in Fig. 5-58 was created using the cabinet and the new phase leg. This test consists of two pulses as shown in Fig. 5-59. The first pulse is large, with the intention of building the current up to the test level. The second pulse is a typical PWM pulse. This pulse is repeated once per second, which makes the power very small while testing large currents. The measured waveform is shown in Fig. 5-60.

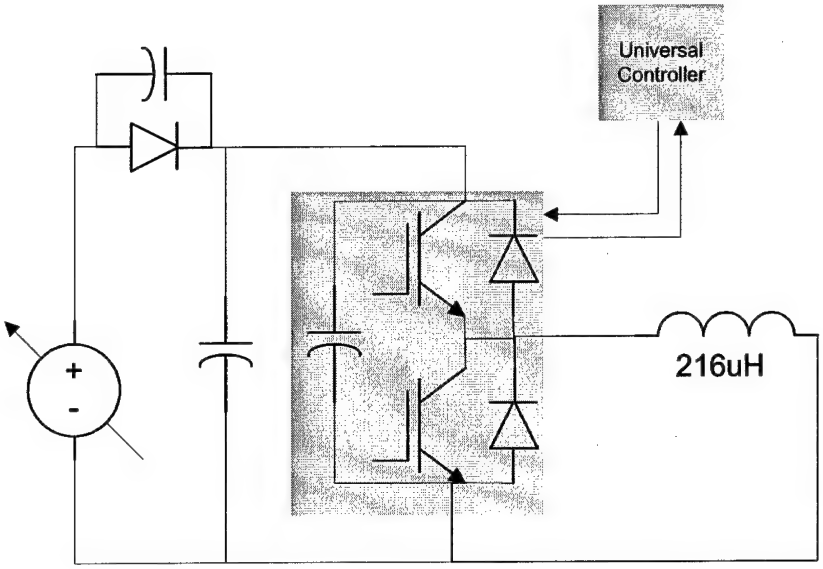


Fig. 5-58 Pulse test schematic using a new PEBB module for EMI verification

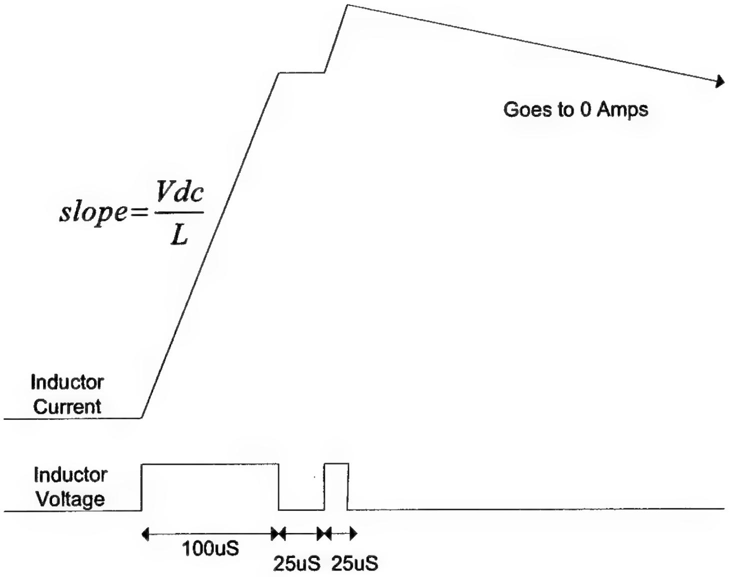


Fig. 5-59 Pulse test applied for EMI verification.

Channel	Description	Scale	Notes
1	VTopSwitch	200V/div	Zero when switch is on
2	VDC	200V/div	
3	Idc	-50A/div	(1div = 10mV), Choke on BNC cable
4	Iload	50A/div	(1div = 10mV), Choke on BNC cable

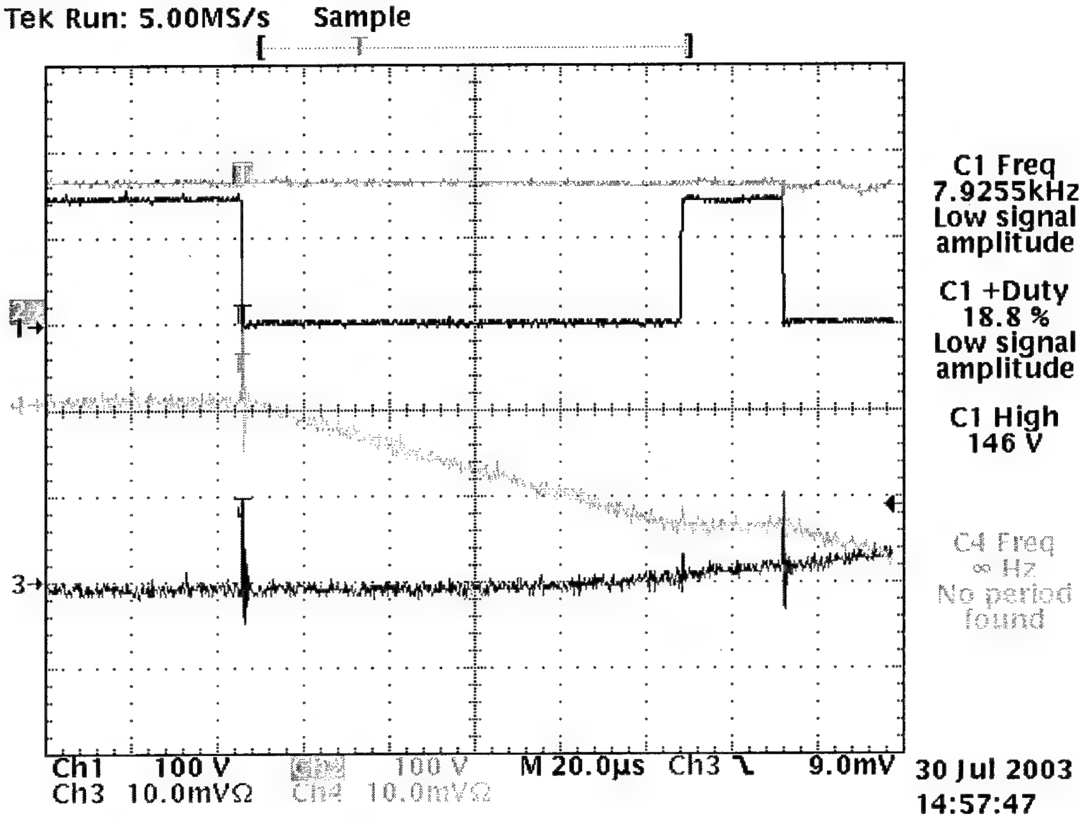


Fig. 5-60 Experimental results showing that the current returns to zero after 12ms.

The current peak can be calculated for turn on:

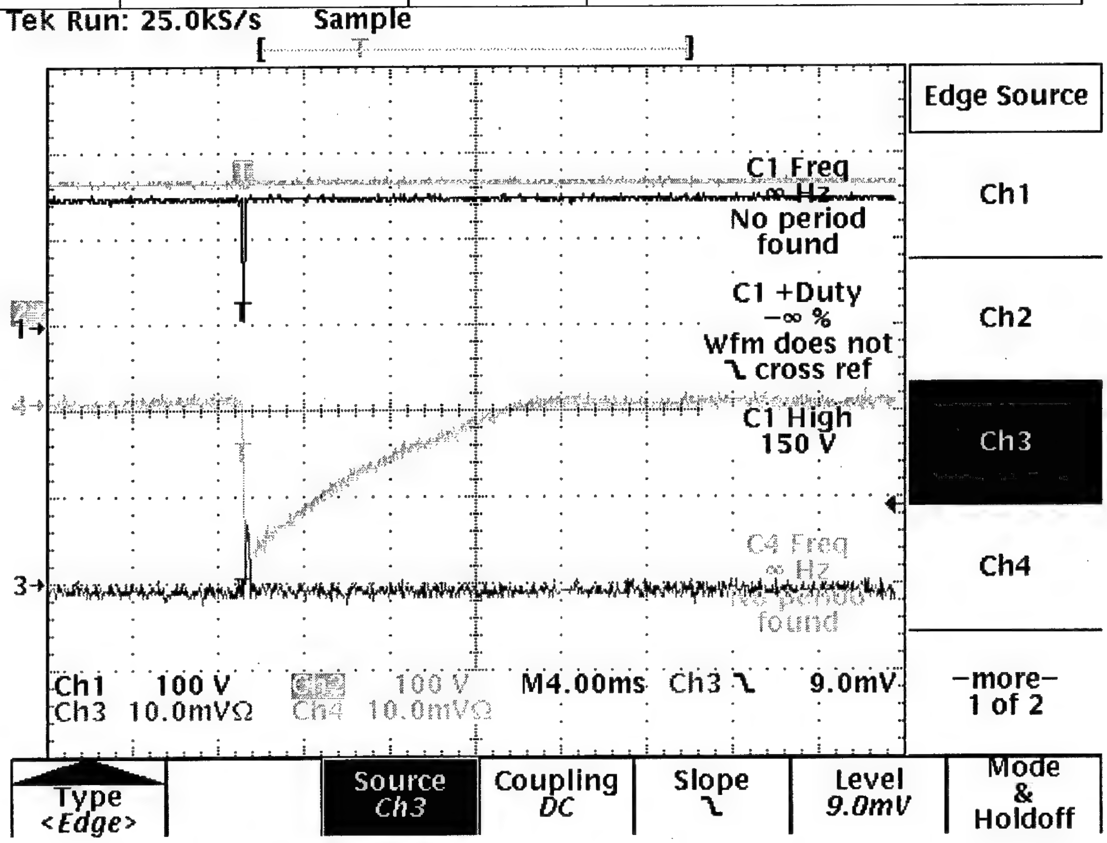
$$I = \frac{VT}{L} = \frac{150 \cdot 100\mu}{216\mu} = 69.4A$$

For turn off, the current peak is:

$$I = \frac{VT}{L} = \frac{150 \cdot 125\mu}{216\mu} = 86.8A$$

A pulse testing interface was written in VHDL to create a waveform similar to that shown in Fig. 5-60 channel 1. This was used in place of the PWM generator, and the values were controlled from the fiber optic interface. A series diode was added to prevent the negative current from flowing back into the power supply. The diode can be seen blocking the negative current in Fig. 5-61.

Channel	Description	Scale	Notes
1	VTopSwitch	200V/div	
2	VDC	200V/div	
3	Idc	-50A/div	(1div = 10mV), Choke on BNC cable
4	Iload	50A/div	(1div = 10mV), Choke on BNC cable



Tek Run: 25.0kS/s Sample

C1 Freq
∞ Hz
No period
found

C1 +Duty
−∞ %
Wfm does not
⤵ cross ref

C1 High
150 V

C4 Freq
∞ Hz
No period
found

Ch1 100 V 100 V M4.00ms Ch3 ⤵ 9.0mV
Ch3 10.0mVΩ Ch4 10.0mVΩ

Fig. 5-61 Experimental test with series diode in dc path to avoid negative currents in the power supply.

Channel	Description	Scale	Notes
1	VTopSwitch	200V/div	
2	VDC	200V/div	
3	Idc	-50A/div	(1div = 10mV), Choke on BNC cable
4	Iload	50A/div	(1div = 10mV), Choke on BNC cable

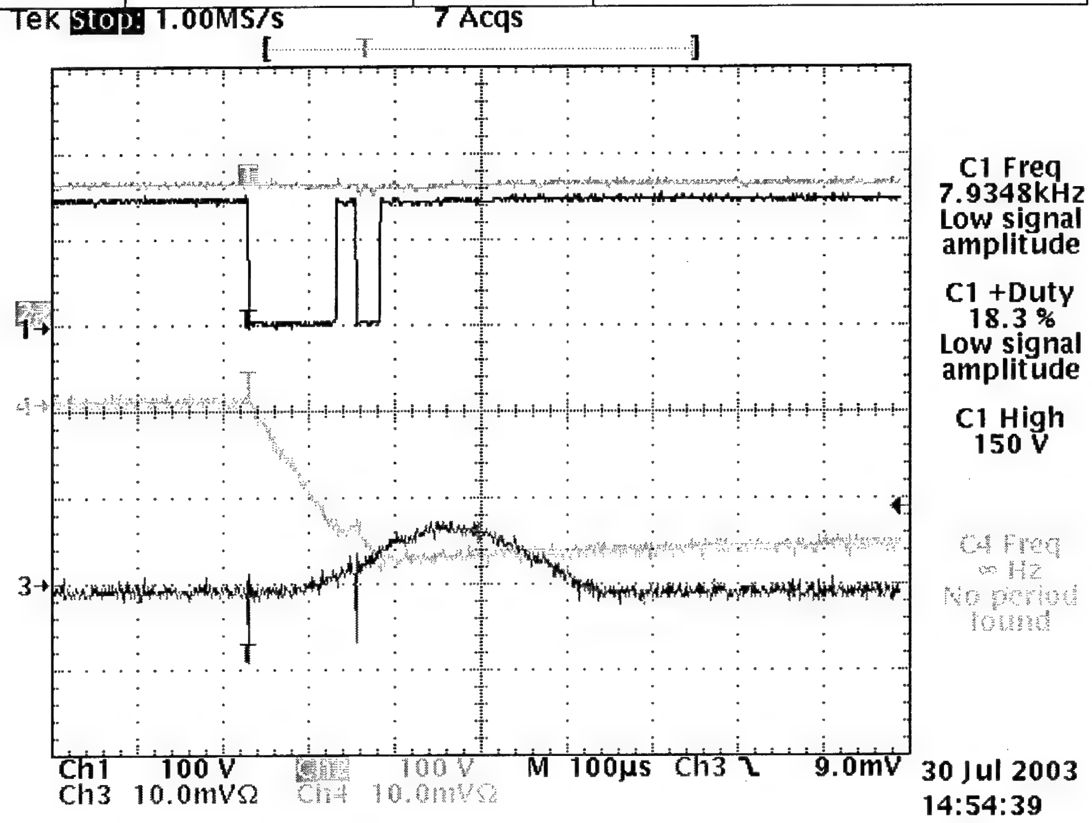


Fig. 5-62 Expanded view of transient in Fig. 5-61.

The following waveforms in Fig. 5-63 show that a peak power of 22.5 kW was achieved at 150A and 150V.

Channel	Description	Scale	Notes
1	VTopSwitch	200V/div	
2	VDiode	10V/div	
3	Idc	-50A/div	(1div = 10mV), Choke on BNC cable
4	Iload	50A/div	(1div = 10mV), Choke on BNC cable

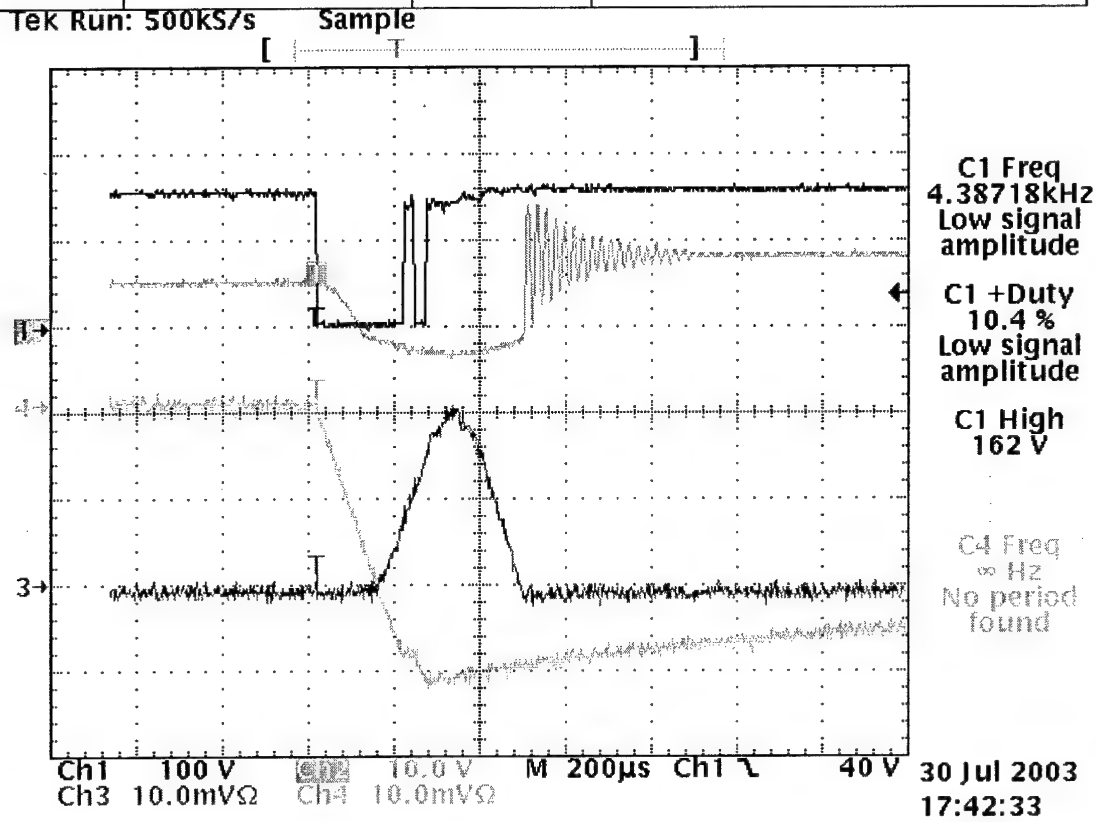


Fig. 5-63 High frequency oscillations observed

5.4.3 Impedance measurements

As shown in the previous results extrinsic oscillations were observed at the commutation instant. In order to study and characterize this phenomenon then impedance measurements using an analyzer were performed. Specifically, the DC impedance, ZDC, was measured using an Agilent 4294A impedance analyzer with the 42941A impedance probe attachment. This is the impedance from the DC positive bus contactor to the DC negative bus contactor. The new phase leg was plugged into slot A having the contactors are open as shown in Fig. 5-64. Fig. 5-65 shows the results obtained. The switching filter can be seen with the peak at 5 kHz, but for high frequencies, the inductance will prevent the filter from working as intended.

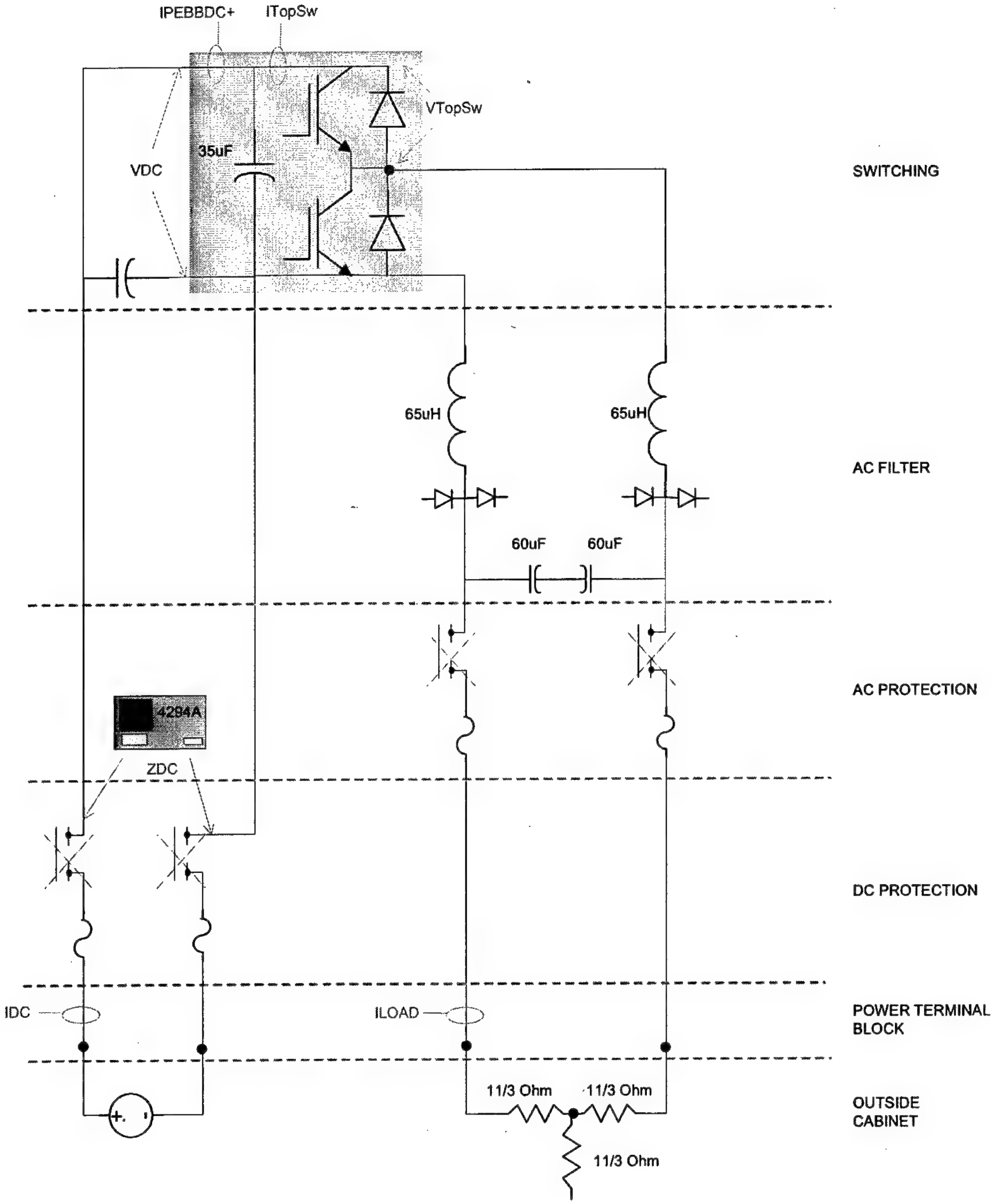


Fig. 5-64 DC Impedance measurement schematic

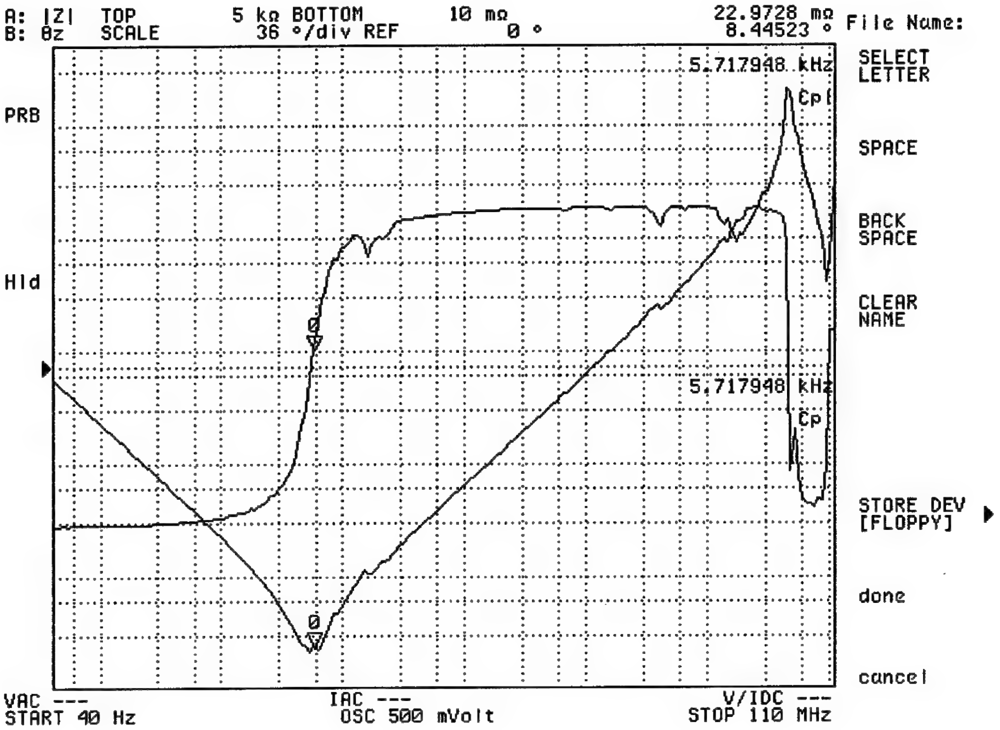


Fig. 5-65 DC Impedance test on the cabinet.

The impedance of the load plus the filter was also characterized, which corresponds to impedance seen by the PEBB module as shown in Fig. 5-66. These results are shown in Fig. 5-67.

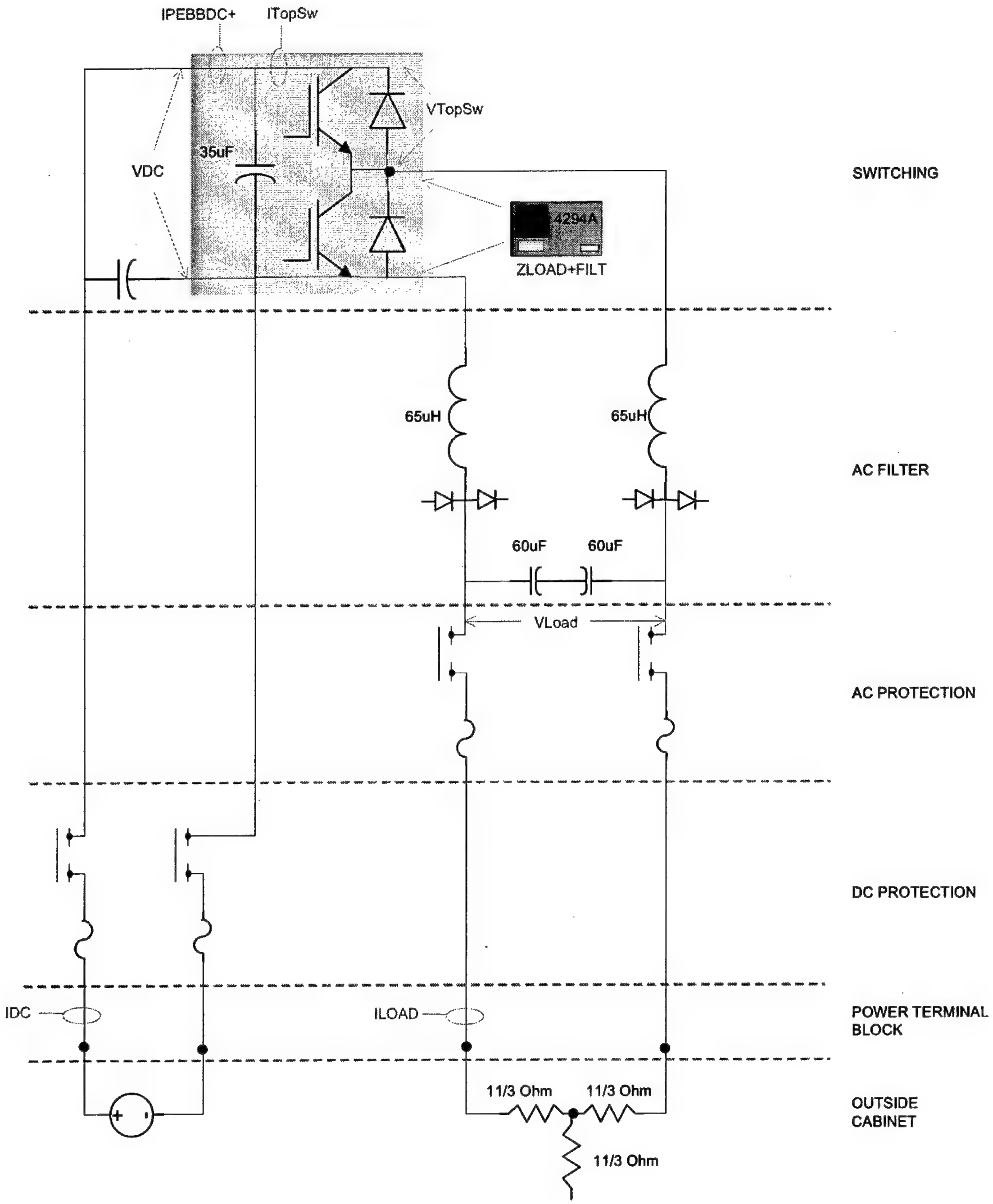


Fig. 5-66 Load impedance measurement setup.

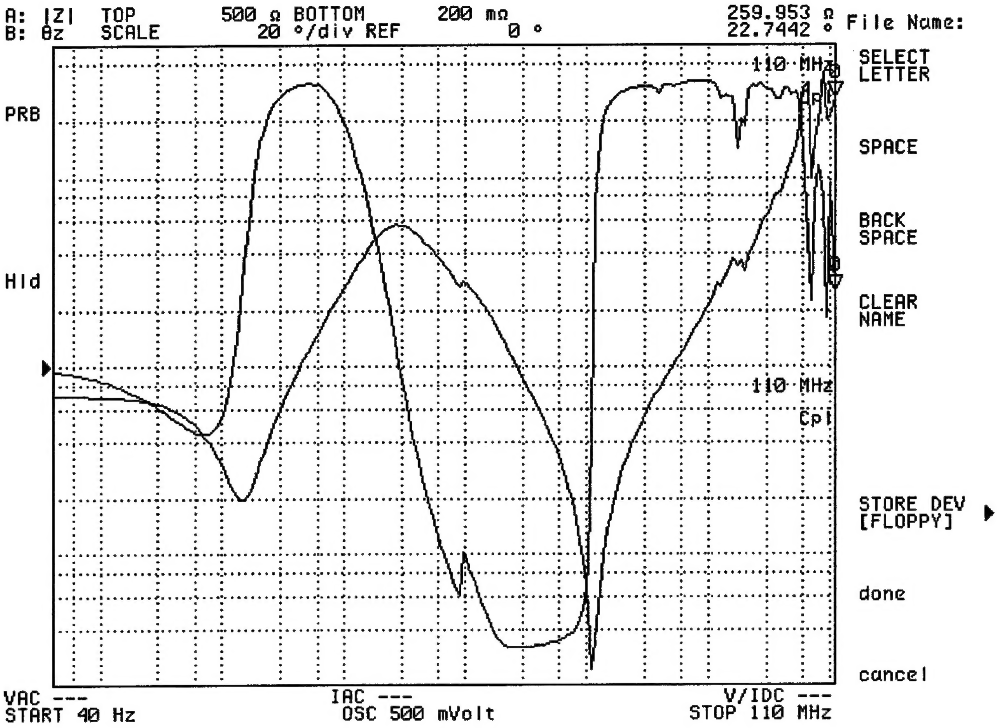


Fig. 5-67 Load impedance measurement results.

5.5 Conclusion

The design of PEBB-based power electronics systems is not only an electrical system design procedure, but has transformed into a multivariable optimization scenario. Physical distributions and structural functionality now play a key role in determining the electrical behavior of the final structure. These basically set parasitic parameters, efficiency, size, volume, reconfiguration capabilities, as well as overall functionality in terms of the PEBB-based power conversion system. Fundamental research in electromagnetic and physical characteristics of such a distributed structure converter is hence required. With an effort on this area good design criteria will be established in conformity with the concepts involved with this novel approach.

6 CONCLUSIONS

This work has investigated an open system design approach for developing Plug and Play PEBB-based power electronics systems. The main objective was to investigate standardized control and communications systems and architectures. The research effort was primarily focused on three distinctive thrusts, namely development of a Universal Controller, development of dataflow-based architecture software for distributed control systems, and development of a Hardware or Power Stage Manager. The main results are summarized as follows.

A Universal Controller board was fully designed, manufactured, and experimentally tested and evaluated. The controller proved its enhanced and powerful computational capabilities attained by its DSP-FPGA based digital system architecture. It also presented an unparalleled flexibility, achieved by the inclusion of JTAG connectors for both DSP and FPGA, of 88 I/O pins connected directly to the FPGA, and its PCI interface. All these provide an outstanding visibility into the controller digital system, greatly simplifying any type of design or evaluations. The board was experimentally verified in all its capabilities by communicating through the upgraded PESNet protocol with similar Universal Controllers, with the new Hardware Manager – performing PWM control over the new PEBB modules-, and also effectively communicating with the previously developed PEBBs upgraded to soft switching capacity. The results obtained have been extremely encouraging, showing a great operational reliability and a significant simplification of the design process. Future results are expected as part of the new project Standard Cell, Open Architecture Power Conversion Systems.

Regarding control software architecture, this project continued with CPES work on embedded control systems proposing a power electronics control software built over standardized ECO. A second version of the control system kernel DARK was finalized and successfully tested through thorough analyses and computer evaluations. In order to study any possible effects of the programming language in the performance of the kernel and hence control system DARK was also implemented using C++, which in general provided a better and more structured way of extending data channels. The PESNet protocol for communications was upgraded and modified adding significant new capabilities, thus increasing the overall reliability of PEBB-based power electronics systems. This protocol was further improved by developing transparent messaging between Universal Controllers across the double-ring fiber optic network. Finally, commercial software platforms for developing embedded control systems were also studied and compared to the proposed dataflow architecture system, where the latter presented more flexible real-time control options, eased the design of distributed control systems, and required significant less redesign efforts. The main disadvantage still remains the lack of a graphical development environment.

On the development of the Hardware Manager, or power stage controller, excellent and encouraging results were attained. In a much shorter design, manufacture, and verification process than the Universal Controller, the Hardware Manager proved its outstanding performance, reliability, and simplicity. Numerous tests actually showed the good performance and operation of the board when communicating through the optic-fiber network with the Universal Controller, and when controlling the new PEBB modules for which it was designed. Electrical and thermal variable readings have been effectively measured and used as part of the control and protection system of the new PEBBs. The success of this board is a direct consequence of all the previous experience gained through the Universal Controller.

Finally, as part of the validation process for the proposed PnP PEBB-based power electronics system a partitioning study was performed to determine the feasible physical, energy, and information boundaries proper to such systems and PEBBs in particular. Correspondingly a PEBB-compliant power stage was designed and built. Previously built 33 kW PEBB modules were upgraded to accommodate for soft switching capability, and new 33 kW PEBB modules using the newly developed Hardware Manager were designed and manufactured. Individual tests with PEBB modules were realized in order to verify their correct operation, both for soft switched and the newly developed ones. Communications tests were also successfully performed between the old and new PEBB modules and the Universal Controller, which effectively controlled these boards. Full verification of the PEBB modules, specifically on the new ones, is still under way in order to ensure their correct operation not only in terms of their functionality but also regarding potential problems such as isolation, thermal, and EMI. Full validation of the different topologies that may be realized with the proposed PnP PEBB-based power electronics systems will be realized as part of the new project Standard Cell, Open Architecture Power Conversion Systems.

REFERENCES

[i]M. Shaw and D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice-Hall, 1996.

[ii]R.Allen and D. Garlan, "A formal approach to software architecture," In Jan van Leeuwen, ed., *Proceedings of IFIP'92.* Elsevier Science Publishers B.V., September 1992.

[iii]A.L. Davis, and R. M. Keller, "Dataflow program graphs", *IEEE Computer,* vol. 15, no.2, Feb, 1982, pp.26-41.

[iv]D. E. Culler, "Dataflow architectures," *Annual Review of Computer Science,* vol. 1, Annual Reviews Inc., Palo Alto, CA, 1986.

[v]B. S. Shuvra, P. K. Murthy, and E.A. Lee *Software synthesis from dataflow graphs,* Kluwer Academic Publishers, Boston ,1996.

[vi]K. Singh, "Design and Evaluation of an Embedded Real-time Micro-kernel," *M.S. thesis,* Department of Computer Science, Virginia Tech, 2003.

[vii] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", Journal of the ACM, Vol. 20, No. 1, pp. 46-61.

[viii]J.J. Labrosse, *MicroC/OS-II, The Real-time Kernel,* R & D Books, Oct. 1998.

[ix]Analog Devices website, http://www.analog.com/.

[x]"Protocol Design of Dual Ring PESNet (DRPESNet)" CPES 2002 Power Electronics Seminar and NSF/Industry Annual Review, April 2002.

[xi]MathWorks Simulink web site, http://www.mathworks.com/products/simulink/.

[xii]MathWorks Real-Time Workshop web site, http://www.mathworks.com/products/rtw/.

[xiii]MathWorks Real-Time Workshop Embedded Coder web site, http://www.mathworks.com/products/rtwembedded.

[xiv]Parool Mody and Stephen H. Edwards, *"Distributed Communication Protocol for Power Electronics System,"CPES Seminar,* Virginia Tech, Blacksburg, VA, 2003

[xv] I. Celanovic, "A Distributed Digital Controller for Power Electronics Systems", M. Sc. Thesis, Virginia Tech, 2000.

[xvi] N. Hingorani, L. Gyugyi, Understanding FACTS: Concepts and Technology of Flexible AC Transmission Systems, IEEE Press, Piscataway NJ, 2000

[xvii] L. Gyugyi et al., "Apparatus and method for dynamic voltage restoration of utility distribution networks", U.S. Patent 5 329 222, July 12, 1994

[xviii] Y. Liang, C. Nwankpa, "A new type of STATCOM based on cascading voltage source inverters with phase-shifted unipolar SPWM", Industry Applications Conference, Thirty-Third IAS Annual Meeting, 1998, Vol.2, pp.1447-1453

[xix] I. Celanovic et al., "A new control architecture for future distributed power electronics systems", Power Electronics Specialists Conference PESC 2000, Vol.1, pp.113-118

[xx] G. . Hua, E. Yang, Y. Jiang, F. C. Lee, "Novel Zero-Current-Transition PWM Converters," in Proc. PESC'93, pp. 538-544.

[xxi]H. Mao, F. C. Lee, X. Zhou, D. Boroyevich, "Improved Zero-Current Transition Converters for High Power Applications," IEEE Trans. Ind. Applicat. Vol. 33, n. 5, 1997.

[xxii] Y. Li, F. C. Lee, J. Lai, D. Boroyevich, "A Novel Three-Phase Zero-Current-Transition and Quasi-Zero-Voltage-Transition (ZCT-QZVT) Inverter/Rectifier with Reduced Stresses on Devices and Components," in Proc. APEC'00, pp. 1030-1036.

[xxiii]Y. Li, F. C. Lee, "A Comparative Study of a Family of Zero-Current-Transition Schemes for Three-Phase Inverter Applications," in Proc. APEC'01, pp. 1158-1164.